Peter Norton

Az IBM PC programozása



Microsoft PRESS

Peter Norton

Az IBM PC programozása

2. kiadás

Peter Norton

Az IBM PC programozása

2. kiadás

Műszaki Könyvkiadó
Budapest, 1992

Az eredeti mű:
PETER NORTON: Programmer's Guide to the IBM PC

Original English language edition Copyright:

© 1985 PETER NORTON

All rights reserved including the right of reproduction in whole or in part in any form.
This edition published by arrangement with the original publisher, Microsoft Press,
a division of Microsoft Corporation, Redmond, Washington, USA.

Lektorálta FÁY ISTVÁN villamosmérnök

© Hungarian translation TÁLLAI BENEDEK és
DR. BÁN PÉTER, 1990, 1992

ETO: 681.3 IBM PC. 06
519.68

ISBN 963 10 8272 5 (1. kiadás)
ISBN 963 10 9421 9

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: SZŰCS PÉTER igazgató
Felelős szerkesztő: BENEDIKTI ISTVÁN
A 2. kiadást sajtó alá rendezte: MOLNÁR ERVIN

AGRO-PRINT Kft. Gyál, 91-241
Felelős vezető: TÓTH ANTAL

Mb. műszaki vezető: DORNIZS LÁSZLÓ
Műszaki szerkesztő: BÁN FERENC
A borítót tervezte: BÁN FERENC
A könyv ábráit rajzolta: PETŐFALVI LÁSZLÓNÉ
A könyv formátuma: B5
Ívterjedelme: 33,625 (A5)
Azonossági száma: 10 073
MŰ: 4499-i-9294
Készült az MSZ 5601–1983 és 5602–1983 szerint
Az 1990-ben megjelent kiadványunk változatlan utánnyomása

Tartalomjegyzék

Köszönetnyilvánítás • 6

Bevezetés • 7

1. A PC anatómiája • 11
2. Kimenet és bemenet • 25
3. A ROM-szoftver • 41
4. Képernyős alapismeretek • 61
5. Lemezes alapismeretek • 87
6. A billentyűzet alapjai • 109
7. Hanggenerálás • 125
8. A ROM-BIOS alapjai • 137
9. A ROM-BIOS képernyős szolgáltatásai • 147
10. A ROM-BIOS lemezes szolgáltatásai • 161
11. A ROM-BIOS billentyűzetszolgáltatásai • 175
12. Egyéb szolgáltatások • 181
13. A ROM-BIOS-szolgáltatások összefoglalása • 197
14. A DOS alapjai • 217
15. DOS-megszakítások • 223
16. Általános DOS-függvények • 241
17. Új DOS-függvények • 265
18. A DOS-szolgáltatások összefoglalása • 295
19. Programépítés • 307
20. A programozási nyelvek • 315

A Melléklet: Installálható berendezésmeghajtók • 341

B Melléklet: Hexadecimális aritmetika • 347

C Melléklet: A karakterkészlet • 355

Köszönetnyilvánítás

Lehetetlenség volna mindenkit felsorolni azok közül, akik hozzájárultak e könyv elkészítéséhez. Mégis van valaki, aki erőfeszítéseivel és elkötelezettségével ezt kiérdemelte:

Suzanne Ropiequet.

Bevezetés

E könyv megírásakor egyszerű, de ambiciózus cél vezérelte a szerzőt: segíteni az IBM személyiszámítógép-család programozásának elsajátítását. Az első IBM személyi számítógép (a PC) bejelentésének pillanatától (1981 őszétől) világos volt, hogy egy rendkívül fontos számítógépet sikerült létrehozni. A PC értékesítése minden várakozást – az IBM-ét is – felülmúlt, és az eredeti modellhez még egy-két rokon is csatlakozott. Idővel a PC-t mindenki elfogadta a professzionális asztali számítógép szabványának. Az eredeti PC-ből egész számítógépcsalád (egy család sok ággal) eredt, ezzel még tovább nőtt a jelentősége.

A PC-család terjedése kiemelte a programfejlesztés fontosságát, amit egyre bonyolultabbá tett az a tény, hogy a család minden tagja (részleteiben és egyes jellemzőiben) eltér a többitől.

Ez a könyv olyan ismereteket, tapasztalatokat és *alapelveket* nyújt, amelyek birtokában a PC-család gépeire készíthetünk programokat, nemcsak a család egyetlen tagja számára, azonban gondunk lesz az egyes családtagok sajátosságaira is. Mindezt oly módon tesszük, hogy programjaink a mostani géptípusokon kívül a következő modelleken is fussanak.

Ezt a könyvet mindenki haszonnal forgathatja, aki programot fejleszt a PC-család gépeire. Programozóknak készült, de *nem csak programozóknak*. Mindenkinek szól tehát, akinek valami oknál fogva meg kell értenie a PC-s programfejlesztés alapelveit és technikai részleteit. Hasznos olvasmány lehet a programozók vezetőjének, a PC-s programok tervezőinek és azoknak a felhasználóknak, akik meg akarják érteni a programozási részleteket is.

Gondolatok a könyvről

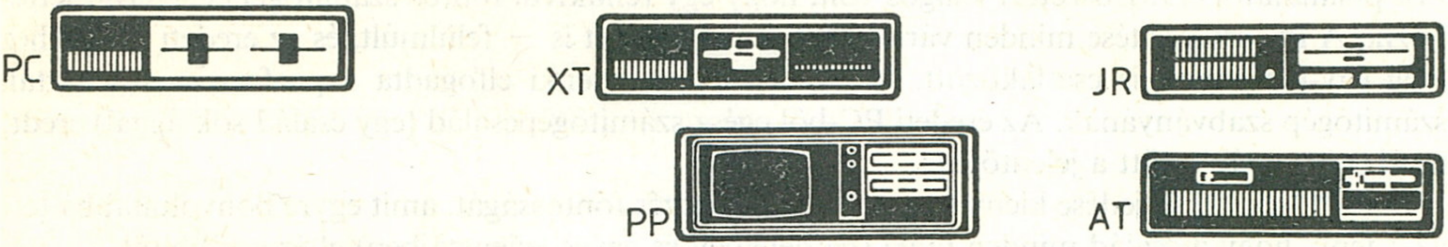
A könyv egyik alapvető fontosságú eleme a programozás *filozófiájának* megtárgyalása. Olvasásakor lépten-nyomon találkozni fogunk azokkal az elvekkel, amelyeket az IBM követett a PC-család tervezésében, ill. amelyeket a PC elmélyültebb programozásának alapelveiként a szerző saját gyakorlata alapján kidolgozott.

Nem teljesítené feladatát ez a könyv akkor, ha csak tényeket közölne, technikai részletek táblázatos formájában. Ez az oka, hogy a technikai ismertetést át- meg átszövik a magyarázatok: mi is az egész PC-család; milyen szálak kötik össze a család egyes tagjait; milyen technikai fogások és módszerek teszik lehetővé, hogy az elkészült programok lépést tartsanak a PC-család fejlődésével.

Hogyan használjuk a könyvet

A könyv egyszerre olvasmány és hivatkozási kézikönyv, ezért legalább kétféle módon közelíthetünk hozzá. Ha úgy tetszik, olvashatjuk az elejétől a végéig (mint bármely könyvet), egy kicsit jobban belemélyedve ott, ahol számunkra hasznos információ van; és gyorsan átlapozva azokat a részeket, amelyekre éppen nincs szükségünk. Ezzel a megközelítéssel átfogó képet alkothatunk a PC-s programok működéséről és ezek elvi hátteréről. Másrészt színtiszta hivatkozási könyvként is használhatjuk úgy, hogy csak egy meghatározott fejezetben mélyedünk el egy bizonyos információ kikeresésére. Minden fejezet elején részletes tartalomjegyzék van, és részletes tárgymutató is található a könyvben, hogy a keresett részt minél könnyebb legyen megtalálni.

Ha a PC programozásának valamilyen részletét akarjuk gyorsan kikeresni, akkor tapasztalni fogjuk, hogy a könyv anyaga szövevényesen összefonódik. Az eligazodást ebben a szövevényben azzal könnyítjük meg, hogy egyes részleteket minden előfordulási helyen megismétlünk, máskor pedig a ☛ szimbólummal utalunk a megfelelő részre (ahogy az adott helyzetben a legcélszerűbb). Ezenkívül a következő szimbólumokkal is segítjük az egyes gépekre specifikusan jellemző anyagok megtalálását:



A gépek sematikus rajzát olyan bekezdések és részek mellett találjuk, amelyek egy bizonyos géptípusra vonatkoznak, míg a géptípus kezdőbetűi az egész gépcsaládra érvényes szövegben az egyes típusokra vonatkozó sajátosságokat jelzik. Reméljük, hogy ezzel megkönnyítjük a keresett információ megtalálását.

Egyéb források

Természetesen egyetlen könyv nem tartalmazhat minden potenciálisan szükséges információt. Mindent megtettünk azért, hogy a könyv a lehető leggazdagabb és legteljesebb legyen, ennek ellenére biztosan szükség lesz további információkra is. Néhány forrás, ahol további kiegészítő információk találhatók:

A részletes technikai információk elsőrendű forrása az IBM Technical Reference kézikönyvsorozata. Külön kézikönyv tartozik az eredeti PC-hez, az XT-hez, a PCjr-hez, az AT-hez és a többi modellhez. A programozáshoz kapcsolódó információ nagyrészt azonos ezekben a könyvekben, és ebben a tekintetben bármelyik kézikönyv megfelel az egész gépcsalád számára. Néhány dolgot szem előtt kell tartani a kézikönyvekkel kapcsolatban. Először: nincs elkülönítve az adott gépre jellemző információ az egész gépcsaládra vonatkozó résztől. A különbségek a józan ész, a kézikönyvek összevetése és a könyv használata segítségével azonosíthatók. Másodszor: figyelembe kell venni, hogy minden új PC, új jellemzőkkel gazdagodik. Ha egy újabb PC-modell kézikönyvét olvassuk, akkor a jellemzők gazdag választékáról találunk adatokat. A régebbi modell kézikönyvében találhatókra támaszkodva elkerülhetjük, hogy olyan lehetőségeket is feltételezzünk, amely nem sajátja a gépcsalád minden tagjának.

Létezik az IBM Options Adapters Technical Reference kézikönyve is, amely különböző opciókat és adaptereket ismertet. (Pl. a PC-családban használatos különböző lemezmeghajtókat vagy megjelenítőket). Ebben a könyvben a berendezésekkel kapcsolatos technikai információk mind megtalálhatók. A kézikönyvet rendszeresen frissítik, az új kiadásra elő lehet fizetni. A benne megtalálható információ nem a programozóknak szól, de egyes részei számukra is hasznosak lehetnek.

Az IBM kiad egy kézikönyvet a bővítésekről is, pl. a PC Networkről (a hálózatról).

Az IBM kézikönyvei közül talán a legfontosabb a DOS-sorozat. Ezek hallatlan bőségben tartalmazzák a részletes technikai információkat – itt ezeket összegeztük. Ha valaki mégis úgy érzi, hogy a DOS működésének mélyebb részleteire van szüksége, akkor ezekhez a kézikönyvekhez kell fordulnia.

Az IBM kézikönyveken kívül egy sereg más forrásban is kutathatunk további információért. Aki bővebben (nem csak a programozásra koncentráló perspektívából) kívánja látni az IBM személyi számítógépet, annak ajánlható a szerző Inside the IBM Personal Computer című könyve. Ehhez hasonlóan a DOS-t mutatja be a Van Wolverton: Running MS-DOS című műve, a Microsoft Press kiadványa. A PCjr sajátosságairól, külső és belső jellegzetességeiről szól a szerző Exploring the PCjr című kiadványa. Az utóbbi is a Microsoft Press kiadásában jelent meg.

Ez a könyv igen átfogóan foglalkozik a PC programozásával, ezért nem várható el (néhány kulcsfontosságú részlettől eltekintve), hogy az egyes programozási nyelvekre is kitérjen. A programozási nyelvekkel és a nyelvek fordítóprogramjaival kapcsolatban túl sok könyvre van szükség, így felsorolásuk vagy ajánlásuk reménytelen lenne.

Ha átolvastuk a bevezető megjegyzéseket, ideje belevetni magunkat a PC-család programozási elveinek elsajátításába!

1 A PC anatómiája

Bevezetés • 12

A 8088 mikroprocesszor • 12

A 80286 mikroprocesszor • 16
A 8087 matematikai társprocesszor • 17

A kiszolgáló áramkörök • 18

A 8259 megszakításvezérlő • 18
A 8237A DMA-vezérlő • 19
A 8284A órajel-generátor • 19
A 8255 programozható perifériaillesztő • 19
A 8253 programozható időzítő • 19
A 6845 CRT-vezérlő • 20
A PD765 hajlékonylemezes vezérlő • 20

A részegységek összekapcsolása: a busz • 20

A címbusz • 21
Az adatbusz • 21

A memória-áramkörök • 21

A tervezés filozófiája • 24

Bevezetés

A programozó szempontjából a PC-család összes tagja processzorból, memóriatokokból és több „okos” vagy programozható áramköri tokból áll. A számítógép főbb áramköri elemei a rendszerkártyán, további lényeges részei pedig a bővítő kártyákon találhatók. Ezek a rendszerkártyába dugaszolhatók.

A rendszerkártyán található a mikroprocesszor (8088 vagy 80286), ehhez kapcsolódik legalább 64 kbájt memória, néhány ROM-ba beépített program – pl. a BASIC és a ROM-BIOS – ill. más fontos kiegészítő áramköri tokok. A tokok közül egyesek külső berendezést vezérelnek, pl. a lemezmeghajtót vagy a megjelenítőt, másokra a mikroprocesszor működéséhez van szükség.

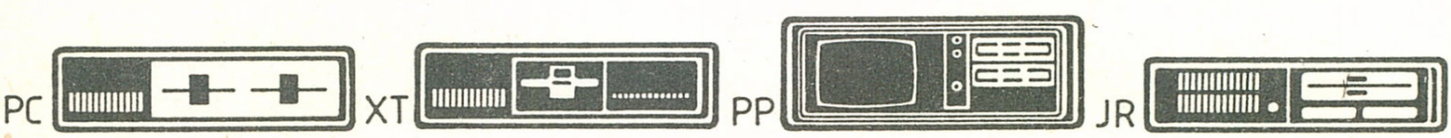
Ebben a részben az összes lényeges áramköri elemmel foglalkozunk, és megadjuk fontosabb technikai jellemzőiket is. Azt is jelezzük, hogy a PC-család melyik tagja használja az áramkört. Szembeötlik a műszaki irodalom tanulmányozásakor, hogy egy áramköri tokot több néven is említenek. Ezek közül az elnevezések közül néhányat mi is megemlítünk majd. Pl. a PC egyes perifériái, köztük a billentyűzet, a 8255 névre hallgató integrált áramkör felügyelete alatt működnek. A név bővebb alakja a 8255A-5, de funkciója alapján nevezik programozható perifériaillesztőnek vagy PPI-nek is. Az említett nevek a PC áramköreinek ugyanazt az alkatrészét jelölik.

A 8088 mikroprocesszor

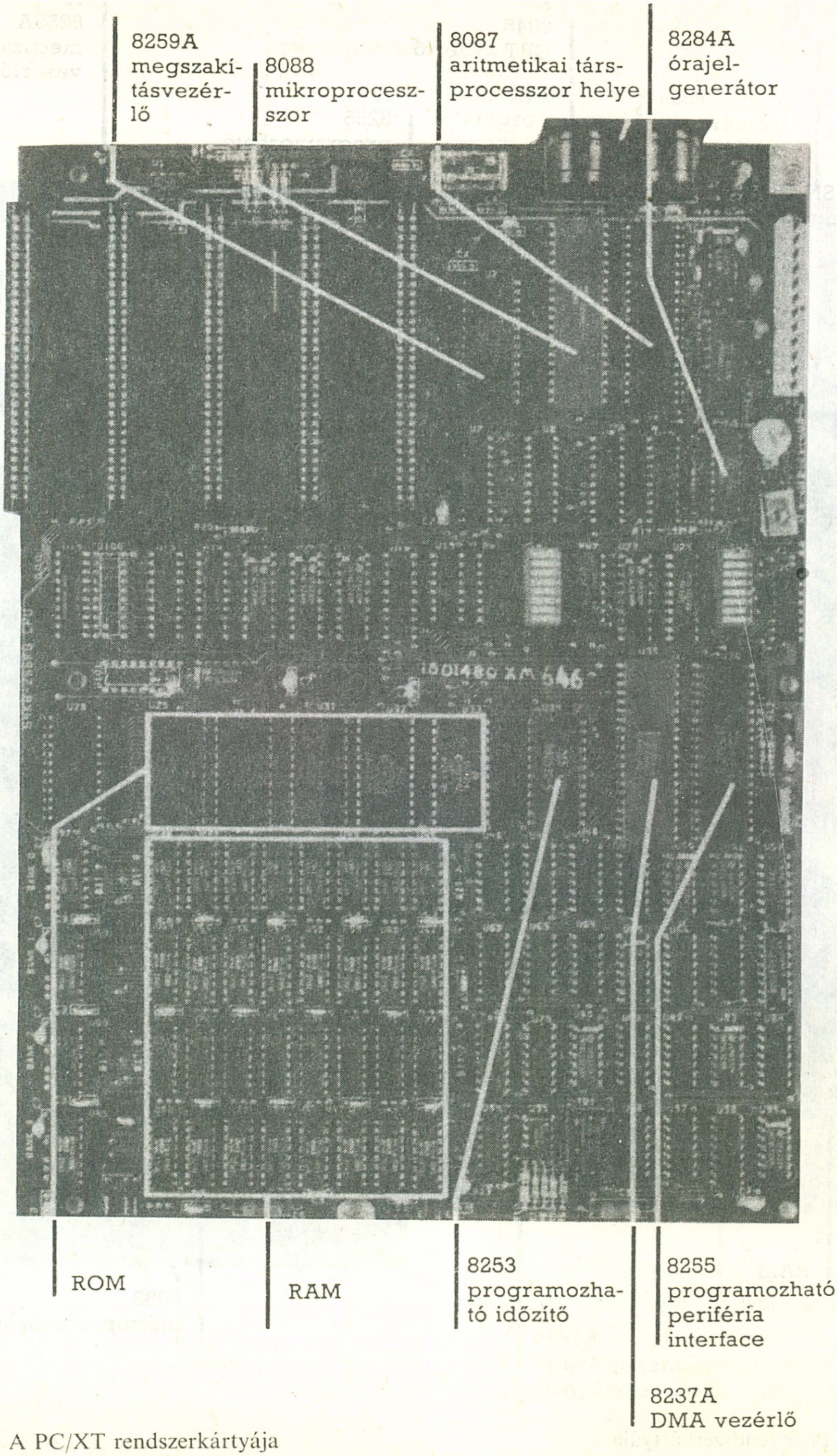
A 8088 mikroprocesszor 16 bites. A szabványos IBM személyi számítógépekben (név szerint az eredeti PC-ben, az XT-ben, a Portable PC-ben és a PCjr-ben) ez a központi egység (CPU, central processing unit) a gép agya. A számítógépbe belépő vagy onnan távozó bitek (feldolgozás vagy átirányítás végett) szinte kivétel nélkül áthaladnak a CPU-n.

A 8088 vezérlőjelek, tárcímek, adatok kiküldésével és vételével irányítja a számítógép alapvető működését. Ezeket a jeleket egy mindent összekötő elektronikus hálózat (az ún. busz vagy más néven sín) mentén továbbítja. A busz mentén helyezkednek el a különböző kiviteli és beviteli portok, amelyek a tároló és a kiszolgáló integrált áramköröket csatolják a buszhoz. Az adatok ezeken a portokon haladnak át a CPU és a gép egyéb részei között.

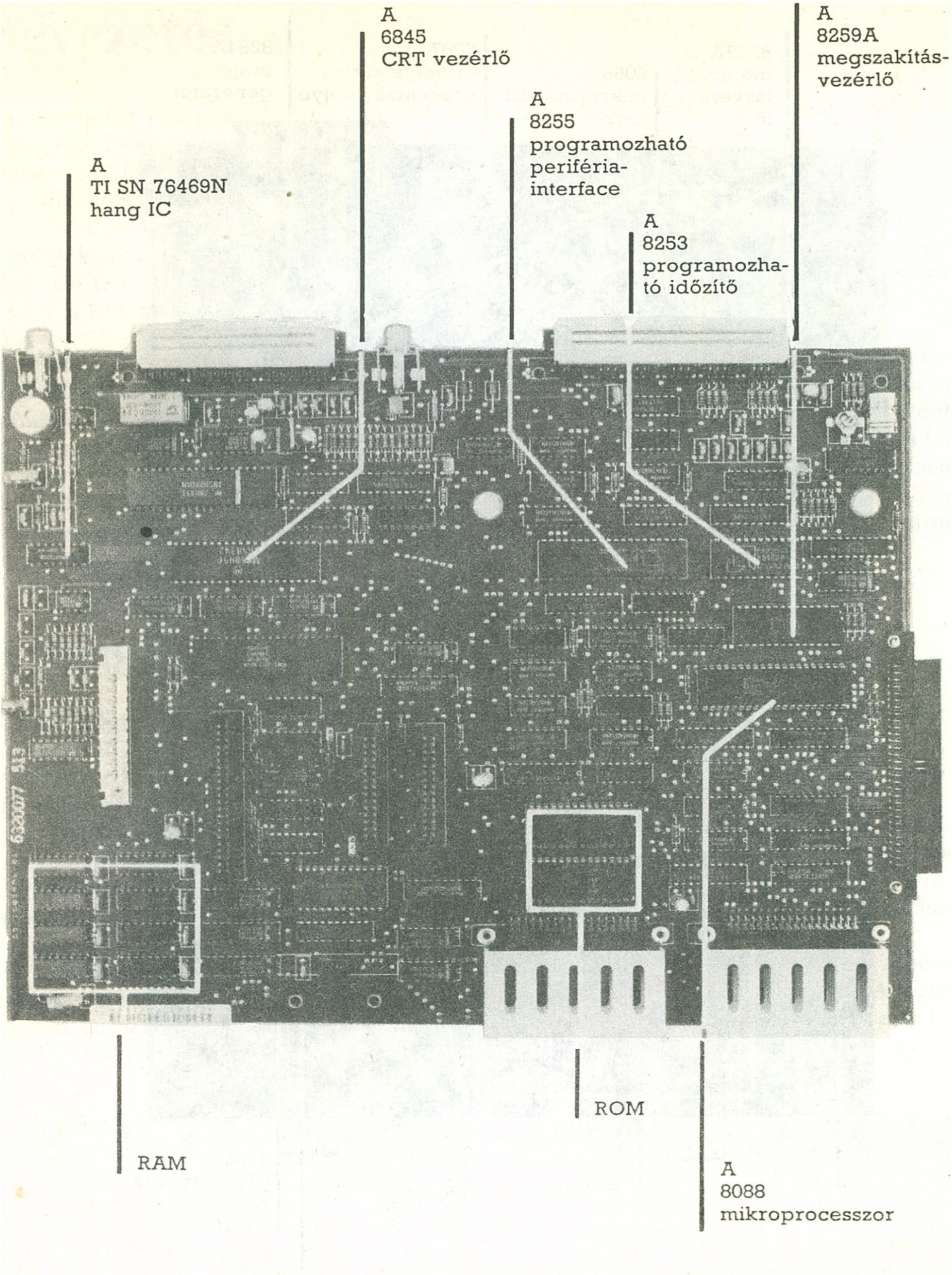
A 8088-on belül 14 regiszter van. Ezek a belső regiszterek 28 bájtot foglalnak el, és képesek adatok, tárcímek, utasításszámlálók, státuszjelek és vezérlőjelek átmeneti tárolására. A regisztereken keresztül a 8088 több, mint 1 millió bájtnyi memóriaelemhez és legföljebb 64 ezer I/O porthoz férhet hozzá. (☛ A 2. fejezetben található a 8088 működésének összefoglalója.)



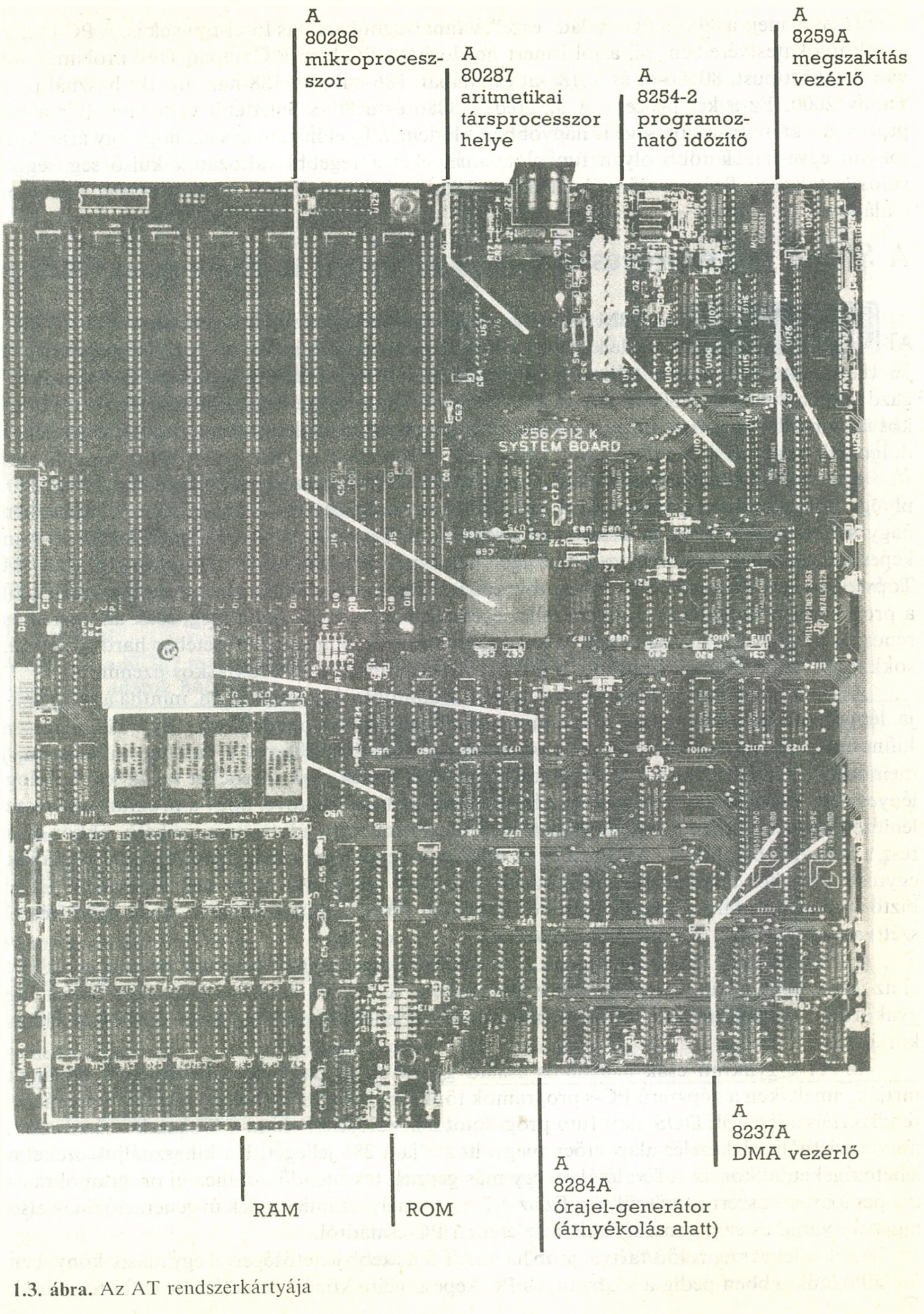
A 8088 családfája. A 8088 csak egyik tagja az Intel 16 bites mikroprocesszor-családjának. A család alapító tagja a 8086. A 8088 csak egy apró részletben tér el a 8086-tól: a 8088 mikroprocesszor 16 bites, adatbusza (a 8086-tal ellentétben) viszont 8 bites. (☛ A 8 bites és a 16 bites busz közötti különbséget l. később, a busz leírásnál.) Gyakorlatilag minden, ami a 8086-ra vonatkozik, érvényes a 8088-ra is: programozási szempontból azonosnak tekinthetjük őket.



1.1. ábra. A PC/XT rendszerkártyája



1.2. ábra. A PCjr rendszerkártyája



1.3. ábra. Az AT rendszerkártyája

Hosszú ideje a 8088 a PC-család „esze”, vannak azonban más Intel-típusok is. A PC-család távoli unokatestvéreiben, pl. a jól ismert hordozható PC-ben, a Compaq Deskproban, 8086 van. Újabb típust, 80186-ot és 80188-at (általában 186-nak, ill. 188-nak hívjuk) használ pl. a Tandy 2000. Ez a két processzor az eredeti 8086 és a 8088 fejlettebb változata. Bár a két processzor számítási sebessége is nagyobb az elődeinél, fő előnye mégis az, hogy egy áramköri tokban egyesítenek több olyan funkciót, amelyeket a régebbi változatok külső segítséggel valósítottak meg. Számos előnyük ellenére a 186 és a 188 nem jelenti az utolsó fázist a 8086-os családban.

A 80286 mikroprocesszor

AT 

Az IBM személyi számítógépek fejlettebb mikroprocesszora a 80286 (röviden 286). Ez a processzor vezérli az AT-t. A 80286 teljes értékű 16 bites mikroprocesszor, teljes 16 bites adatbusszal, és újabb programozási vonásokkal gazdagítja a 8086-ot. A 80286-nak talán legnagyobb újdonsága, hogy lehetővé teszi a többtaskos üzemmódot és a virtuális tárkezelést. Ez a két fogalom azok előtt ismert, akik nagy gépen dolgoztak már.

A többtaskos üzemmód azt jelenti, hogy a CPU képes egyszerre több feladatot futtatni – pl. egy dokumentum nyomtatásával egy időben egy táblázatkezelő programot –, miközben nagy sebességgel váltogatja a futtatott programot. Egy szokványos PC igen korlátozottan képes a többtaskos üzemmódra. Ehhez igen kifinomult programokra van szükség: pl. az IBM Topview vagy a Microsoft Windows programjára. Egy valódi többtaskos processzor (pl. a 286) a programok váltogatását csaknem teljes egészében saját maga oldja meg, csak az operációs rendszertől igényel némi segítséget. Mivel a többtaskos üzemmód a 286 esetében hardver szintű, sokkal gyorsabb és megbízhatóbb, mint a szoftverrel megoldott többtaskos üzemmód.

A virtuális tárkezelés lehetővé teszi, hogy a számítógép úgy viselkedjen, mintha a memóriája lényegesen nagyobb lenne, mint amennyivel ténylegesen rendelkezik. Egy különlegesen kifinomult hardver és szoftver révén a programmal elhitethető, hogy 1 Gbájt (egy milliárd bájt) memóriával rendelkezik, bár a tényleges áramkörök ennek csak töredékét teszik ki. A dolog lényege egy pontosan kidolgozott címzési rendszer, ez teszi lehetővé, hogy a program egy része lemezen legyen. Ha nem a memóriában lévő utasításra vagy adatra van szükség, akkor ez a rész a lemezről betöltődik. A 286-ra és az operációs rendszerre nehéz feladat hárul: kitalálják egyrészt azt, hol található az információ; másrészt pedig azt, hova kell kimenteni. Ezzel biztosítják a program zökkenőmentes és hatékony futását, annak ellenére, hogy egyes részei szét vannak szórva a számítógéprendszerben.

A virtuális tárat régóta használják a különböző típusú számítógépeken, de csak most jött el az ideje, hogy a mikrók világában is bevezessék. A 286 alkalmazása igen nagy hatást fog gyakorolni a programokra, mivel a programok terjedelmét csak a lemez tárolókapacitása korlátozza.

Az AT-t gyakran csak a 8086-os család gyorsabb és nagyobb teljesítményű tagjának tartják, amelyiken a népszerű PC-s programok többsége futtatható (beleértve a DOS operációs rendszert és a legtöbb DOS alatt futó programot is). Valójában mind a többtaskos üzemmód, mind a virtuális tárkezelés alapvetően megváltoztatja a 286 jellegét. Ha kihasználjuk ezeket a lehetőségeket, akkor az AT valójában egy más gépnek tekintendő, amihez új programokra és új operációs rendszerre van szükség. Ez az AT-t a személyi számítógépek új generációjának első tagjává avatja, és ez a generáció eltér az eredeti PC-családtól.

Az előzőeket szem előtt tartva, jobb ha az AT fejlettebb lehetőségeivel egy másik könyvben foglalkozunk, ebben pedig a szabványos PC képességeire koncentrálunk. Az is fel fog tűnni,

hogy a könyvben tárgyalt programozási technikák zöme a 8088-ra koncentrál, és a megfelelő helyen külön kitérünk a 80286-ra.

A 8086 család minden tagját úgy tervezték, hogy kiegészítő processzorokkal működjön együtt. Képesek együttműködni két speciális társprocesszorral is: a 8087 matematikai és a 8089 I/O társprocesszorral. Ezek az opcionális áramkörök átvehetik a CPU munkájának egy részét. Az IBM csak a 8087-hez szükséges bővítő áramköröket támogatja, ezért most ezt az áramkört tárgyaljuk részletesebben.

A 8087 matematikai társprocesszor

A 8088 csak egész számokkal (röviden egészekkel) tud műveletet végezni. A „valós" vagy lebegőpontos számok kezelése különleges eszközöket igényel. Ez rendszerint szubrutinokat jelent, amelyek ugyan megfelelően elvégzik a lebegőpontos műveleteket, ez azonban a hatékonyság és a sebesség rovására megy.

A 8087 társprocesszorral a lebegőpontos számítások 10-szer, 50-szer gyorsabban végezhetők el, mint a 8088-cal. Sőt, az aritmetikai műveletek pontossága is jelentősen megnő, a 8088-hoz vagy a legtöbb drága, sok millióba kerülő nagygéphez viszonyítva. A 8087 az egyszerű összeadáson, kivonáson, szorzáson, osztáson kívül képes trigonometriai számításokra is (szinusz, koszinusz, tangens stb.). Ez sok bonyolult programozási feladatot leegyszerűsít. Az eddigieken kívül a számábrázolásban végezhet műveletet egész, lebegőpontos, sőt decimális alakú számokkal is. Ráadásul mindezt azalatt végzi el, amíg a 8088 mással van elfoglalva.

1.1 táblázat

A 8087 és a 80287 matematikai társprocesszorok 8 darab 80 bites regiszterében kezelt numerikus adatok határai

Adattípus	Közelítő intervallum		Értékes számjegyek	
	(tól)	(ig)	Bitek	Decimális
Szavas egész	−32 768	+32 767	16	4
Rövid egész	−2 × 10E9	+2 × 10E9	32	9
Hosszú egész	−9 × 10E18	+9 × 10E18	64	18
Pakolt decimális	−99...99	+99...99	80	18
Rövid valós	8,43 × 10E−37	3,37 × 10E38	32	6−7
Hosszú valós	4,19 × 10E−307	1,67 × 10E308	64	15−16
Átmeneti egész	3,4 × 10E−4932	1,2 × 10E4932	80	19

Minden 8088 alapú PC-modellt előkészítettek a 8087 befogadására – a PCjr kivételével. Igaz, kell szoftvertámogatás is. ([AT] Az AT a 80287 társprocesszort használja, amit a 80286 igényeihez igazítottak.) Bár a 8087 nagyon megnöveli az IBM személyi számítógép aritmetikai teljesítményét, mégis csak viszonylag kevés program használja. Ez a szomorú helyzet egy egyszerű történelmi tény következménye: bár a 8087 felhasználásának lehetőségét tekintetbe vették az eredeti PC – és a legtöbb modell – tervezésekor, az IBM nem támogatta a 8087-et, sőt tudomást sem vett potenciális hasznáról egészen addig, míg ki nem alakultak a PC hardver

és szoftver szabályai. Ez azt jelentette hogy a PC-re kifejlesztett eredeti hardver és szoftver nagy része nem számolt a 8087 felhasználásának lehetőségével, tekintélyes számítási kapacitástól fosztva meg ezzel mindnyájunkat.

Bár ezidáig nem túl sok 8087-et értékesítettek, egyre több olyan programmal lehet találkozni – mint pl. az Ashton-Tate Frameworkje –, amelyik érzékeli a 8087 jelenlétét; és a program igényeitől függően vagy használja, vagy megkerüli a használatát. Sajnos, szinte egy kézen meg lehet számolni a kapható programokat.

Mivel a PC-családon belül ritkán használják a matematikai társprocesszort, nem foglalkozunk programozásának sajátosságaival.

A kiszolgáló áramkörök

A mikroprocesszor nem képes egymagában irányítani az egész számítógépet, de erre nincs is szükség. Ha bizonyos irányítási feladatokat más integrált áramkörökre bíz, akkor a CPU a saját feladataival foglalkozhat. Ezeknek a kisegítő áramköröknek az lehet a feladatuk, hogy a teljes belső hálózaton irányítsák az információ áramlását, pl. a megszakításvezérlő és a DMA-vezérlő (Direct Memory Acess). Más kiszolgáló áramkörök a számítógép és a hozzá kapcsolt külső berendezés közötti információ áramlását irányítják (pl. a képernyőét vagy a lemezmeghajtóét). Ezek az ún. *berendezésvezérlők* (device controller) gyakran külön kártyán találhatók, és a PC egyik bővítő csatlakozójához kell dugaszolni őket.

Az IBM PC kiszolgáló IC-inek jó része programozható. Ez azt jelenti, hogy be lehet őket állítani bizonyos speciális feladatok ellátására. Legtöbbször nincs sok értelme az IC-k közvetlen programozásának, de az egyes IC-k most következő leírásában kiemeljük, hogy melyiket lehet biztonsággal átprogramozni, és melyiket nem. Mivel ez a könyv nem foglalkozik a hardver közvetlen irányításával, az egyes IC-k programozásának részleteit az IBM kézikönyvében kell megkeresni.

A 8259 megszakításvezérlő

A 8259 a megszakítások működését felügyeli. A megszakítási jelet a hardver küldi a CPU-nak, vagy azért, hogy felhívja magára a figyelmet, vagy valamilyen intézkedés kiváltása céljából. A 8259 fogadja ezeket a jeleket, és meghatározza fontosságukat a korábban érkezett jelekhez viszonyítva. A döntés alapján küld megszakítást a CPU-nak. Amint a CPU érzékeli a megszakítási jelet, elindítja a megadott berendezéshez tartozó programot. Végeredményben ez a program hajtja végre a szükséges műveletet. (☛ A megszakításokat részletesebben tárgyaljuk a 2. és a 3. fejezetben.)

A 8259 egyszerre 8 megszakítást tud kezelni, de több 8259 összekapcsolásával ez bővíthető.

AT Az IBM kihasználta ezt a bővítési lehetőséget az AT-ben. Két 8259-et kapcsolt össze, így az egyidejűleg 15 megszakítást kezelhet.

Általában nem szokás a 8259-et átprogramozni, mivel az valószínűleg alapvetően megzavarná a gép működését. A fő program futása közben azonban bármikor lehetőség van a prioritási szintek megváltoztatására. Ez azt jelenti, hogy egy program a saját igényeinek megfelelően megváltoztathatja a megszakításkérések feldolgozásának sorrendjét.

A megszakításvezérlőt szokás még 8259A-nak, INTR-nek és PIC-nek nevezni. Az utóbbi programozható megszakításvezérlőt (Programmable Interrupt Controller) jelent.

A 8237A DMA-vezérlő

Annak érdekében, hogy a CPU-t megkíméljük a fölösleges „zaklatástól", a számítógép egyes részei a CPU megkerülésével közvetlenül fordulhatnak a memóriához. Ezt a műveletet közvetlen memória-hozzáférésnek, DMA-nak nevezzük, és a 8237A nevű IC végzi. A DMA-vezérlő legfőbb feladata, hogy a lemezes írási és olvasási műveleteket a mikroprocesszor nélkül is végre lehessen hajtani. Mivel a lemezes I/O műveletek meglehetősen lassúak, a DMA jócskán felgyorsítja a számítógép működését.

A PC-család valamennyi tagja a (PCjr kivételével) 8237A-t vagy ezzel egyenértékű áramkört használ a közvetlen memória-hozzáférés megvalósításakor. [JR] Részben a DMA hiánya okozza, hogy a PCjr lassúbb az unokatestvéreinél. A DMA-vezérlő segítsége híján a PCjr 8088-asának minden lemezes műveletre figyelnie kell. Közvetve ez az oka, hogy a Junior billentyűi nem használhatók lemezművelet közben.

A DMA-vezérlőben négy külön csatornán lehet az adatokat a memóriába, ill. a memóriából továbbítani, és 344 bites saját memóriában lehet az áthaladó adatokat tárolni. Több DMA-vezérlő is összekapcsolható, [AT] az AT-ben is valójában kettő van.

A 8284A órajel-generátor

Az órajel-generátor szolgáltatja a mikroprocesszor és a perifériák működéséhez szükséges többfázisú órajelet. Alapfrekvenciája 14,3128 MHz. A többi IC rendszerint ennek a frekvenciának valamilyen egész számmal osztott részét használja. A szabványos PC-család 8088-asa 4,77 MHz-cel működik, ez az alapfrekvencia egyharmada. A 8086 család újabb tagjainak nagyobb a frekvenciája. (A PC egyes változataiban használt 8088-2 processzor 8 MHz-es frekvenciával működik, és ez szinte megkétszerezi a 8088 számítási teljesítményét). A 80286 processzor 6 MHz-cel működik, ez kb. ötven százalékos többletet jelent a 8088-hoz képest. A belső busz és a 8253 programozható időzítő (☛ hamarosan sorra kerül) 1,193 MHz-cel működik, ez a 8088-énak egynegyede, az alapfrekvenciának pedig tizenkettedrésze.

A 8255 programozható perifériaillesztő

A 8255 feladata, hogy a számítógép egyes perifériáit a buszhoz kapcsolja. Az egyes berendezésekhez futó vagy onnan érkező adatok (pl. hangszóró vagy kazettás magnetofon) ezen az IC-n keresztül jutnak el az I/O porthoz.

A 8255-öt szokás még 8255A-5-nek vagy PPI-nek (Programmable Peripheral Interface, programozható perifériaillesztő) nevezni. Általában a rendszerszoftver feladata a PPI beprogramozása, ezért bár lehetséges, de nem szükséges az IC programozása.

A 8253 programozható időzítő

A 8253 ([AT] az AT-ben 8254A) többcélú időzítő, számláló. Programmal beállítható háromféle, pontos késleltetési idő. A szükséges jeleket a 8284A órajel-generátorból kapja. Működési frekvenciája 1,190 MHz.

A 8253-at főleg arra használják, hogy a PC beépített hangszóróját működtesse, de más frekvenciafüggő feladatokra is felhasználják (pl. a kazetta I/O műveletei és az idő nyilvántartá-

sa). (☛ A 7. fejezetben a hangképzés tárgyalásakor további adatok találhatók erről az IC-ről.)

A 8253-at szokták 8253-5-nek (vagy időzítőnek) és néha órának is nevezni. Mindig ügyeljünk arra, hogy „órának" nevezik a 8284-es IC-t, amelyik a számítógép 14,3 MHz-s alapfrekvenciáját szolgáltatja.

A 6845 CRT-vezérlő

A 6845 – más néven a Motorola CRT IC – általában egy bővíthető kártyán, az ún. videoadapteren található. Az IC 19 belső regisztere segítségével programozható és vezérelhető a raszteres képernyő. Bár magunk is programozhatnánk, okosabb ezt a PC BIOS-ára bízni. (☛ a 4. fejezetben további adatok találhatók a megjelenítőkről és az adapterekről.)

A PD765 hajlékonylemezes vezérlő

A PD765 irányítja és felügyeli a hajlékonylemezes meghajtó működését. Gyakrabban FDC-nek (floppy-disk controller) vagy NEC-vezérlőnek (Nippon Electric Company) is nevezik. Hasonlóan a 6845 CRT-vezérlőhöz, tanácsos a BIOS-ra bízni a kezelését.

A részegységek összekapcsolása: a busz

Mint már korábban említettük, a PC-család belső vezérlő áramköreit a busz köti össze. A busz nem más, mint a fő áramköri kártyán lévő közösen használt vezetékrendszer, amelyre rácsatlakoznak a számítógép irányító komponensei. Ezen haladnak az adatok az egyik részegységből a másikba.

A PC minden vezérlő IC-je – és a tár minden egyes bájtja – közvetlen vagy közvetett módon a buszra csatlakozik. Mikor egy új kártyát csatlakoztatunk egy bővítő csatlakozóba, akkor valójában közvetlenül a buszra csatlakoztatunk. Ezzel az új kártya az egész gép működésének egyenrangú tagja lett.

Minden gépbe kerülő vagy azt elhagyó információ legalább átmenetileg a busz mentén tárolódik. Az adat leggyakrabban a memóriába kerül, ez a PC-családban 8 bites memóriacellák ezreit jelenti. Előfordul azonban, hogy egyes adatok rövid időre porton vagy regiszterben kötnek ki, arra várakozva, hogy a CPU a megfelelő helyre küldje őket. A portok és regiszterek egyszerre általában csak egy vagy két bájtot tárolnak, és csak átmeneti állomásai az egyik helyről a másikra tartó adatnak. (☛ A portokat és a regisztereket a 2. fejezetben tárgyaljuk részletesen.)

Ha egy memóriacellát vagy portot tárolási célra használunk, akkor egy egyértelmű címmel azonosítjuk. Amikor az adat továbbításra kész, a rendeltetési hely címét először el kell küldeni a címbuszon, és ezt követi az adat az adatbuszon. A busz tehát magánál az adatnál többet továbbít. Teljesítményt és vezérlőinformációt is hordoz, pl. időzítőjeleket (a rendszerórától), megszakításjeleket, valamint ezernyi memóriacellának, ill. a buszhoz kapcsolódó berendezéseknek a címét. Ezek teljesítésére a busz a következő négy részre oszlik: a *tápvonalak*, a *vezérlőbusz*, a *címbusz* és az *adatbusz*. A címbusz és az adatbusz rejtelmeibe ásunk le mélyebben, mert az adattovábbítás módjának tanulmányozása a PC-család néhány egyedülálló sajátosságának megértéséhez is hozzásegít.

A címbusz

A szabványos PC-családban a címbusz 20 vezetéken továbbítja a tárcellák és a buszra csatlakozó berendezések címét. (☛ A tárcímzést részletesen ismertetjük a memóriaáramkörök leírásában és a 3. fejezetben.) Mivel minden egyes vezetéken két érték lehetséges (1 vagy 0), a szabványos PC 2^{20} címet tud megkülönböztetni. Ez több, mint egymillió címet jelent. [AT] Az AT-nek 24 címvonala van, ezzel 2^{24}, vagyis több, mint 16 millió cím adható meg.

Az adatbusz

Az adatbusz a címbusszal együtt működik, hogy adatokat továbbítson a számítógépen belül. A PC 8088-ra alapozott rendszere nyolcvonalas adatbuszt használt, ezeknek mindegyike egy bináris számjegyet (bitet) továbbít. Tehát a nyolcvonalas buszon az adatok 8 bites (bájtos) egységekben haladnak. [AT] Az AT 80286-os mikroprocesszora 16 bites adatbuszt használ, tehát az adatokat 16 bites (szavas) egységekben továbbítja.

Mivel a 8088 mikroprocesszor 16 bites, egyidejűleg 16 adatbittel dolgozhat, akárcsak rokona, a 80286. Bár a 8088 belül 16 bites számokkal tud dolgozni, az adatbusz mérete miatt 8 bites egységekben továbbítja az adatokat a környező áramkörök felé. Ez egyeseket arra a megjegyzésre késztetett, hogy a 8088 nem valódi 16 bites mikroprocesszor. Maradjunk annyiban, hogy mégis az – bár a 80286-nál kisebb teljesítményű.

A 80286 16 bites adatbusza valóban hozzájárul a gyorsabb adatátvitelhez, de az XT és az AT közötti sebességkülönbség fő oka az AT nagyobb órafrekvenciájából és hatékonyabb belső szervezéséből ered.

A gazdaságosság miatt sok számítógép – köztük a PC-család régebbi tagjai – a 8088-at és a hozzátartozó 8 bites adatbuszt használja a 8086 és a 16 bites adatbusz helyett. Nagyon sok 8 bites áramköri elem van, széles a választékuk és olcsók. A PC tervezésekor a 16 bites áramkörök még sokkal drágábbak és nehezen beszerezhetők voltak. A 8088 alkalmazása nemcsak azért volt lényeges, hogy a PC ára alacsony legyen, hanem segített elkerülni az alkatrészhiányt is. Azóta a 16 bites áramköri elemek ára jóval olcsóbb lett, ezért gazdaságossá vált a 80286-nak és 16 bites adatbuszának alkalmazása. A 286 arra is képes, hogy vegyesen használjon 8 bites és 16 bites alkatrészeket, ezzel biztosítva a kompatibilitást a PC-családon belül.

A memória-áramkörök

Az eddigiekben foglalkoztunk a CPU-val, a kiszolgáló áramkörökkel és a busszal, de éppencsak érintettük a memóriát. A memória tárgyalását a fejezet végére hagytuk, mert a memória-IC-k az eddigi IC-től eltérnek. Nem irányítják a számítógépen belüli információáramlást, csak tárolják.

A számítógépen belül található memória-IC-k száma határozza meg a programok és adatok számára a tárterület nagyságát. A gépenkénti eltérésektől eltekintve egy PC-ben általában 40 kbájt csak olvasható memória (ROM, read only memory) – bővítési lehetőséggel –, és 128 kbájt-tól 256 kbájt-ig terjedő írható és olvasható memória (RAM, random access memory) van. Mivel a rendszerkártyán csak meghatározott (pl. 256 kbájt vagy 512 kbájt) RAM-nak van hely, a PC-hez a bővítő csatlakozóin keresztül különböző kapacitású bővítőt csatlakoztathatunk. Ez azonban nem egyéb, mint a szabványos PC memóriájának fizikai képe.

A számítógép számára a memória-IC-k csak több ezer 8 bites (bájtos) tárolócellát jelentenek, amelyiknek mindnek egyedi címe van. A programozó számára is ezt jelenti. Nem arra kell gondolni, hogy fizikailag mekkora a memória, hanem arra, hogy mekkora a megcímezhető. A 8088 processzor 1024K vagy 1 048 576 bájt tárat tud megcímezni. Valójában ez a címek maximális száma, tehát legföljebb ennyi különböző memóriacellára hivatkozhat. (☛ A memória címzését részletesen a 2. fejezetben ismertetjük.)

Minden egyes bájtra egy 20 bites számmal (mint címmel) hivatkozunk. A 8088 memóriarendszerében a címek azért 20 bit „szélesek", mert egy 20 bites címbuszon haladnak. A tárrekeszek címét általában hexadecimálisan jelöljük, nem binárisan; a 20 bites címet tehát 5 hexadecimális számjegyre írjuk át. Ennek segítségével 00000-tól FFFFF-ig terjedő címtartomány adható meg (tizes számrendszerben 0-tól 1 048 576-ig.) (☛ Akinek nehézséget okoz a hexadecimális számrendszer, annak a *B* függelék segít.)

Amikor a PC 1024 kbájt-os címezhető memóriájáról beszélünk, általában 16, egyenként 64 kbájt-os blokkra osztjuk. Az egyes blokkokat az első hexadecimális számjeggyel azonosítjuk, vagyis a blokk összes címében közös legmagasabb helyiértékű résszel. A memória első 64 kbájt-ja a 0. blokk, ebben a bájtok címe 00000-től 0FFFF-ig tart. Az utolsó blokk az F. blokk, ebben találhatók az F0000 – FFFFF címek.

1.2 táblázat

A PC-család tárblokkjainak vázlata

F 0 0 0 0	Állandó ROM-terület: ROM-BIOS, BASIC, diagnosztika
E 0 0 0 0	Cserélhető ROM-terület
D 0 0 0 0	Cserélhető ROM-terület
C 0 0 0 0	BIOS-bővítések (XT-lemez)
B 0 0 0 0	Hagyományos képernyőterület (PC, XT és AT)
A 0 0 0 0	Képernyőtár bővítése
9 0 0 0 0	Munka RAM 640K-ig
8 0 0 0 0	Munka RAM 576K-ig
7 0 0 0 0	Munka RAM 512K-ig
6 0 0 0 0	Munka RAM 448K-ig
5 0 0 0 0	Munka RAM 384K-ig
4 0 0 0 0	Munka RAM 320K-ig
3 0 0 0 0	Munka RAM 256K-ig
2 0 0 0 0	Munka RAM 192K-ig
1 0 0 0 0	Munka RAM 128K-ig; a legnagyobb lehetséges PCjr-en
0 0 0 0 0	Munka RAM 64K-ig; a rendszerszoftver szokta használni.

Szinte minden szempontból igaz, hogy nincs merev határ a blokkok között. Részben kényelmi okok miatt hivatkozunk a blokkokon belüli memóriára, részben pedig azért, mert minden IBM személyi számítógépben a blokkhatárokon változik a memóriához rendelt funkció.

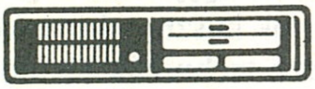
Elméletileg minden tárterületen lehet állandó ROM, vagy változtatható tartalmú RAM. Az első 10 blokkot (0-tól 9-ig), összesen 640K-t, a RAM számára tartják fenn az IBM PC-ben, és közönséges munkaterületként használják. A gépbe épített memóriát mindig erre a területre teszik, az első blokktól kezdve. Mivel a RAM-nak folytonosan ki kell töltenie a tárat, egy blokk

sem maradhat ki. Ha ezen a 640 kbájt-os tartományon belül nagyobb címre hivatkozunk, mint amennyi memória ténylegesen van, az rendkívüli helyzetet teremt. Ha a program ilyen címet próbál meg használni, akkor az eredmény változó lehet, bár általában semmi rendkívüli sem látszik: a program fut tovább.

Az összes IBM személyi számítógépnél legalább 64K memória van a 0. blokkban. Természetesen a 64K a minimális tár, és ritkaságnak számít, ha egy PC-ben csak ennyi van. A 0. blokk legalacsonyabb címeit hagyományosan a rendszerprogramok számára tartják fenn. Ezeken a címeken státuszinformációt, címtáblázatokat, karaktertáblázatokat és az operációs rendszer szubrutinjait tárolja a gép. (☛ A 3. fejezetben részletesen ismertetjük az alacsony címek tartalmát.)

Az A. blokk a videomemória bővítésére fenntartott terület, amelyet az IBM javított grafikus adaptere (EGA, Enchanced Graphics Adapter) és a professzionális grafikus adapter (PGA, Professional Graphics Adapter) használ. Az A. blokk használata igen sajátságos és szokatlanul cseles – olyannyira, hogy kevés hasznos és megbízható adatot tudunk közölni róla. Legjobb, ha az A. blokkot olyan átmeneti jegyzetfüzetnek tekintjük, amelyet a bonyolultabb videomegoldások használnak néha.

A B. blokkot közönséges videomemóriának használja az összes modell, a PCjr kivételével. Két 32K-s részre van felosztva, ezek a B0000 és a B8000 címen kezdődnek. Az egyszerűség kedvéért ezekre a területekre csak mint B0-ra és B8-ra hivatkozunk. Az IBM monokrom adapter – ez a kártya vezérli az IBM monokrom monitort – a B0 terület első 4K-ját használja (a további 28K kihasználatlan). Az IBM színes – grafikus adapter – ez a kártya vezérli a legtöbb egyéb monitort – a B8 terület első 16K-ját használja (a maradék 16K kihasználatlan).

JR 

A PC-modellekben általában választható az adapter típusa, sőt használhatják mind a kettőt. A PCjr-be azonban csak egy a színes-grafikus adapternek megfelelő adapter van beépítve, monokrom adapter nem is használható. A PCjr a B8 használatát imitálja, de valójában a RAM magasabb című részét használja (a 0. blokkot a PCjr alapmodellje, az 1. blokkot pedig a 128K-s bővített modell) videoadatokhoz. Egy speciális áramkör, az ún. videokaputömb sebességével a Junior pontosan utánozza a PC videofunkcióját. A programok látszatra a B. blokkot, a PC szokásos videomemóriáját használják. A PCjr tehát úgy viselkedik, mint egy színes-grafikus adapterrel felszerelt PC. Szerencsére a PCjr olyan jól leplezi a közte és a PC-család többi tagja közt fennálló különbségeket, hogy programozói szempontból eltekinthetünk a Junior sajátosságaitól, és szokásos PC-ként kezelhetjük.

Nagyon fontos szempont, hogy a B. blokk felhasználási módja és annak felosztása a B0 monokrom és a B8 színes-grafikus részre teljesen általános szabvány a PC-családban. Minden PC-modell – a PCjr-t is beleértve – és a PC minden képernyőadaptere (köztük az EGA és a Professzionális Adapter) a szabványos B. blokkos tárolót használja, vagy úgy tesz, mintha azt használná.

A C. blokk a jelenlegi ROM-programok esetleges bővítésére van fenntartva. Az IBM első ízben az XT fix-lemezének kezeléséhez szükséges programok számára használta fel ezt a területet (a lemezzel a PC is bővíthető). A szubrutinokat nem a C. blokk elejére vagy végére tették (ahogyan ezt az ember elvárná), hanem a közepére, a C8-tól kezdve. Talán jogos a feltételezés, hogy a BIOS legtöbb bővítése erre az általános rendeltetésű területre kerül, különösen azok, amelyek az új hardverbővítések kiszolgálásához szükségesek.

JR A D. és az E. blokk a PCjr-nél bevezetett cserélhető ROM számára van fenntartva. Szinte minden PC-modellhez használhatnánk cserélhető ROM-ot, de a Junior kivételével ritkán használják fel. A cserélhető memória valójában vagy a blokkok elejére, vagy a közepére kerül: D0-tól, D8-tól, E0-tól vagy E8-tól. Kizárólag a PCjr-ben kerülhet a következő blokkba is, F0-tól vagy F8-tól.

Általában az F. blokkot a mindig készenlétben álló ROM-programok számára használják. Ide tartozik a ROM „kazettás" BASIC, a ROM-BIOS, valamint a tesztelő és diagnosztizáló szubrutinok. (☛ További részletek a 3. fejezetben.) JR A PCjr különleges célra használja az F. blokkot: ha az F. blokkba memóriát helyezünk, akkor az „felülírja" a szokásos ROM-BIOS-programokat. (☛ A 3. fejezetben további részletek találhatók a Juniorban alkalmazott cserélhető memóriáról.)

A tervezés filozófiája

Mielőtt elkezdenénk a következő fejezeteket, meg kell tárgyalnunk a PC-család tervezését vezérlő elveket. Így könnyebb lesz megérteni, hogy mi fontos és hasznos – vagy épp lényegtelen és haszontalan – a számunkra.

Az IBM személyi számítógép tervezésének részét képezi a BIOS kiszolgáló szubrutinjainak gyűjteménye. (☛ L. 8-tól 13-ig fejezet.) Lényegében ezek tartalmazzák az IBM által fontosnak tartott összes vezérlőfunkciót és -műveletet. A PC-család alapfilozófiája: hagyjuk a dolgot a BIOS-ra, ne vesződjünk a közvetlen vezérléssel. Ez igen jó ötlet, számos hasznos következménnyel. A BIOS-szubrutinok használata elősegíti a helyes programozási technikák alkalmazását, és segít elkerülni azokat a kusza trükköket, amely oly sok más számítógép átka volt. Növeli annak az esélyét is, hogy a programok a PC-család minden tagján működni fognak. Ezenkívül nagyobb rugalmasságot biztosít az IBM számára, hogy javítsa és bővítse a PC-számítógépek sorát. Ugyanakkor naivitás lenne egyszerűen kijelenteni: „Ne vesződjünk a hardver közvetlen kezelésével". Előfordulhat – indokoltan vagy indokolatlanul, passzióból vagy kényszerből –, hogy a programban a lehető legközvetlenebb módon akarjuk kezelni a hardvert. Ezt nevezik szemléletesen úgy, hogy *programozás a csupasz vas szintjén.*

Ha programunkkal közvetlenül akarjuk kezelni a hardvert, akkor ezt kizárólag csak a portokon keresztül tehetjük. (☛ Részletesen l. a 2. fejezetben). Kivéve, ha az adatokat a képernyőre küldjük – ezt a tárba írással érjük el. A PC hardverjének közvetlen működtetése abból áll, hogy az adatokat hardver portokon küldjük ki. A portok felhasználása a PC közvetlen irányítására (néhány kivételtől eltekintve) ellenkezik az IBM tervezési elveivel, és mi is újfent ennek az elkerülését tanácsoljuk. A szabály alól csak az jelent kivételt, ahol az IBM nem biztosította a BIOS-eszközöket. Különleges esete ennek a hanggenerálás (☛ l. a 7. fejezetet).

2 A kimenet és a bemenet

Bevezetés • 26

Hogyan kommunikál a 8088 • 28

A 8088 adatformái • 28

Hogyan címezi a 8088 a tárat • 29

A tár kibővítése szegmentált címzéssel • 29
A 8088 14 regisztere • 30

Hogyan használja a 8088 a portokat • 38

Családon belüli különbségek a portok használatában • 38

Hogyan használja a 8088 a megszakításokat • 40

Bevezetés

Általában elmondható, hogy minél többet tud valaki a programozásról, annál inkább kezdi felismerni a programozási nyelvek korlátait. A magasszintű programozási nyelveket (pl. a BASIC-et vagy a C-t) nem úgy tervezték, hogy a programozás minden problémájára megoldást adjanak – bár vannak köztük jobbak és rosszabbak is. Elérkezhet az a pillanat, hogy a rendszer mélyebb szintjére kell leszállni, mert olyan szubrutint akarunk használni, amit maga a nyelv használ. Előfordulhat, hogy még mélyebbre szeretnénk hatolni, és a hardver szintjén akarunk programozni.

Annak ellenére, hogy néhány nyelvben közvetlenül belenyúlhatunk a tárba (pl. a BASIC-ben a **PEEK** és a **POKE**), vagy közvetlenül is kommunikálhatunk a portokkal (ugyancsak a BASIC-ben az **INP** és az **OUT**), a legtöbb programozó végül mégis az assembly nyelv segítségére szorul. Az assembly nyelv az alap, amire a többi programozási nyelv épül. A többi assembly nyelvhez hasonlóan a 8088 assembly nyelve is szimbolikus utasításkódok gyűjteménye (l. a 2.1. táblázatot). A 8088 belsejében ezek a kódok és a hozzájuk tartozó adatok bináris formára fordulnak le: ezt nevezzük gépi nyelvnek. Ebben a formában a memóriában tárolhatjuk őket, és az elektronikus áramkörökön áthaladva mindegyik elvégzi a maga feladatát.

Azokat a műveleteket, amelyek a 8088 utasításai elvégeznek, néhány kategóriába sorolhatjuk. Az aritmetika a négy alapműveletet tartalmazza 8 és 16 bites számokkal. Vannak adatmozgató utasítások. Képesek arra, hogy egyedi biteket kezeljenek – nem is túl ügyetlenül. Meg lehet vizsgálni bizonyos értékeket, és az eredménytől függően logikai döntés hozható. Végül, de nem utolsó sorban, képesek együttműködni a környező áramkörökkel. Az utasítások mérete 1 bájt és 6 bájt között változhat. A tervezés eredményeként a legalapvetőbb és leggyakrabban használt utasítások a legrövidebbek.

Az assembly nyelvű programozásnak két szintje van: az egyik szint az illesztőrutinok szintje, amelyen a magasszintű nyelv és az alacsonyabb szintű DOS vagy ROM-BIOS-szubrutinok közötti kapcsolatot valósítják meg. A másik szint teljes egészében assembly program, amelyen programunk egészen egzotikus feladatokat is végrehajthat a hardver szintjén, ezzel olyan hőstettre is képes, mint senki más. Bármelyik utat is választjuk, ha meg akarjuk érteni az assembly nyelv használatát, előbb meg kell ismerkedni vele: hogyan kezeli a 8088 az adatokat, és hogy működik együtt a gép többi részével. A fejezet hátralévő részében elsősorban azzal foglalkozunk, hogyan kommunikál a 8088 a számítógép többi részével.

2.1. táblázat

A 8088 utasításkészlete

A mnemonik kód	Az utasítás neve	A mnemonik kód	Az utasítás neve
AAA	ASCII-igazítás összeadáshoz	CBW	Bájt szóvá konvertálása
AAD	ASCII-igazítás osztáshoz	CLC	Az átviteljelző nullázása
AAM	ASCII-igazítás szorzáshoz	CLD	Az irányjelző nullázása
AAS	ASCII-igazítás kivonáshoz	CLI	A megszakításjelző nullázása
ADC	Összeadás átvitellel	CMC	Az átviteljelző komplementálása
ADD	Összeadás	CMP	Összehasonlítás
AND	Logikai ÉS	CMPS	Bájt vagy szó összehasonlítása (lánc)
CALL	Szubrutinhívás	CMPSB	Bájtlánc összehasonlítása

A mnemonik kód	Az utasítás neve	A mnemonik kód	Az utasítás neve
CMPSW	Szólánc összehasonlítása	LEA	Tényleges cím betöltése
CWD	Szó konvertálása dupla szóvá	LES	Mutató *ES*-be töltése
DAA	Decimális igazítás összeadáshoz	LOCK	A busz blokkolása
DAS	Decimális igazítás kivonáshoz	LODS	Bájt vagy szó betöltése (lánc)
DEC	Dekrementálás	LODSB	Bájt betöltése (lánc)
DIV	Osztás	LODSW	Szó töltése (lánc)
ESC	Tárrekesz elérése	LOOP	Ciklus
HLT	Állj!	LOOPE	Ciklus, amíg egyenlő
IDIV	Egészek osztása	LOOPNE	Ciklus, amíg nem egyenlő
IMUL	Egészek szorzása	LOOPNZ	Ciklus, amíg nem nulla
IN	Bájt vagy szó beolvasása	LOOPZ	Ciklus, amíg nulla
INC	Inkrementálás	MOV	Mozgatás
INT	Megszakítás	MOVS	Bájt vagy szó mozgatása (lánc)
INTO	Megszakítás túlcsordulásnál	MOVSB	Bájt mozgatása (lánc)
IRET	Visszatérés megszakításból	MOVSW	Szó mozgatása (lánc)
JA	Ugrás, ha több	MUL	Szorzás
JAE	Ugrás, ha több vagy egyenlő	NEG	Negálás
JB	Ugrás, ha kevesebb	NOP	Üres utasítás
JBE	Ugrás, ha kevesebb vagy egyenlő	NOT	Logikai NEM
JC	Ugrás, ha átvitel van	OR	Logikai VAGY
JCXC	Ugrás, ha a *CX* nulla	OUT	Szó vagy bájt kiírása
JE	Ugrás, ha egyenlő	POP	Kivétel a veremből
JG	Ugrás, ha nagyobb	POPF	A jelzők kivétele a veremből
JGE	Ugrás, ha nagyobb vagy egyenlő	PUSH	Verembe töltés
JL	Ugrás, ha kisebb	PUSHF	A jelzők verembe töltése
JLE	Ugrás, ha kisebb vagy egyenlő	RCL	Forgatás az átviteljelzőn át balra
JMP	Ugrás	RCR	Forgatás az átviteljelzőn át jobbra
JNA	Ugrás, ha nem több	REP	Ismétlés
JNAE	Ugrás, ha nem több vagy egyenlő	RET	Visszatérés
JNB	Ugrás, ha nem kevesebb	ROL	Forgatás balra
JNBE	Ugrás, ha nem kevesebb vagy egyenlő	ROR	Forgatás jobbra
JNC	Ugrás, ha nincs átvitel	SAHF	Az *AH* betöltése a jelzőkre
JNE	Ugrás, ha nem egyenlő	SAL	Aritmetikai léptetés balra
JNG	Ugrás, ha nem nagyobb	SAR	Aritmetikai léptetés jobbra
JNGE	Ugrás, ha nem nagyobb vagy egyenlő	SBB	Kivonás átvitellel
JNL	Ugrás, ha nem kisebb	SCAS	Bájt vagy szó átvizsgálása (lánc)
JNLE	Ugrás, ha nem kisebb vagy egyenlő	SCASB	Bájt átvizsgálása (lánc)
JNO	Ugrás, ha nincs túlcsordulás	SCASW	Szó átvizsgálása (lánc)
JNP	Ugrás, ha a paritás nulla	SHL	Léptetés balra
JNS	Ugrás, ha az előjel nulla	SHR	Léptetés jobbra
JNZ	Ugrás, ha nem nulla	STC	Az átviteljelző 1-re állítása
JO	Ugrás, túlcsordulásnál	STD	Az irányjelző 1-re állítása
JP	Ugrás, ha a paritás 1	STI	A megszakításjelző 1-re állítása
JPE	Ugrás páros paritásnál	STOS	Bájt vagy szó tárolása (lánc)
JPO	Ugrás páratlan paritásnál	STOSB	Bájt tárolása (lánc)
JS	Ugrás, ha az előjel 1	STOSW	Szó tárolása (lánc)
JZ	Ugrás nullánál	SUB	Kivonás
LAHF	A jelzők betöltése *AH*-ba	TEST	Vizsgálat
LDS	Mutató *DS*-be töltése	WAIT	Várakozás
		XCHG	Csere
		XLAT	Átkódolás
		XOR	Logikai kizáró VAGY

Hogyan kommunikál a 8088

A 8088 háromféleképpen érintkezik az őt körülvevő áramkörvilággal: közvetlen vagy közvetett memória-hozzáféréssel, portokon keresztül és a megszakításnak nevezett jelekkel.

A *tár*at úgy használja, hogy adatokat olvas ki vagy ír be a számszerű címmel azonosított tárrekeszből. A tárrekeszhez kétféle módon lehet hozzáférni: közvetlenül – a 8237A IC-n keresztül (ezt nevezik közvetlen memória vagy DMA-vezérlőnek), vagy közvetve – a 8088 belső regiszterein keresztül. A lemezmeghajtók és a soros kommunikációs portok közvetlenül elérhetik a tárat a DMA-vezérlőn keresztül. Az összes többi berendezés a 8088 regiszterein keresztül tart kapcsolatot a tárral. (☛ A DMA-ról további részletek találhatók az 1. fejezetben. A regiszterekre még visszatérünk ebben a fejezetben.)

A tártól eltekintve a 8088 alapvetően *port*ok segítségével kommunikál a számítógéppel. A tárhoz hasonlóan a portokat is számokkal azonosítják, és adat írható be, ill. olvasható ki belőlük. A portok kiosztása egy bizonyos számítógépnél egyedi. Általánosságban igaz, hogy a portok kiosztása az IBM PC-családján belül azonos, csak némi eltérés van a modellek között. (☛ L. a 2.4. táblázatot is.)

Megszakításokkal jelzi a külvilág a 8088-nak, hogy valami történt (pl. leütöttek egy billentyűt), és kéri a megfelelő válaszintézkedést. Bár a 8088 és a külvilág kapcsolata szempontjából a megszakítások létfontosságúak, de más célra is használhatók. Pl. szoftver megszakításokkal kérheti a rendszer-BIOS vagy az operációs rendszer a speciális kiszolgáló programok végrehajtását. A megszakítások igen fontosak a PC programozásában, ezért a fejezet végén külön részben foglalkozunk vele.

A 8088 adatformái

Numerikus adat. A 8088 csak 4 egyszerű, egész típusú adattal tud műveletet végezni. Az adatforma két alapelemre épül: a 8 bites bájtra és a 16 bites (2 bájtos) szóra. Mindkét alapvető egység a 16 bites műveletvégzésből és a 8 bites adatbuszból ered. A kettő közül a bájt az alapvetőbb, és amikor a 8088 a tárat megcímezi, egy bájtnyi egységet címez. Egyetlen bájtban 0 és 255 közötti előjeltelen egész lehet (2^8 lehetőség). Ha az adat előjeles, értéke -128 és $+127$ közé eshet. (☛ L. a 2.2. táblázatot.)

Ha egybájtosnál nagyobb egész értékekre van szükség, akkor a 8088 egyszerűen két szomszédos bájtot használ, és egy egységnek tekinti őket. A 2 bájtos szó a legáltalánosabb

2.2. táblázat

A 8088 négy adattípusa

Méret	Előjel	Tartomány	
		Decimális	Hexadecimális
8	Nincs	0-tól 255-ig	00-tól FF-ig
8	Van	– 128-tól + 127-ig	80-tól 7F-ig
16	Nincs	0-tól 65 535-ig	0000-tól FFFF-ig
16	Van	– 32 768-tól + 32 767-ig	8000-tól 7FFF-ig

forma. Ha a 2 bájtos szót előjeltelen egésznek tekintjük, akkor értéke 0 és 65 535 közé eshet. Előjeles számként az érték −32 768 és +32 767 közé esik.

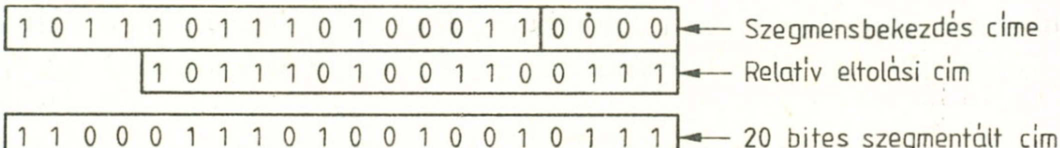
Karakteres adat. A karakteres adatot a szokásos ASCII formában tárolja, minden egyes karakter 1 bájtot foglal el. A 8088 semmit sem tud az ASCII-karakterekről, közönséges bájtként kezeli őket, egy nevezetes kivétellel: a 8088 képes összeadni és kivonni ASCII numerikus karakterekkel ábrázolt számokat. A tényleges műveletet binárisan végzi, de az *AF* feltételjelző és néhány speciális utasítás segítségével hatékonyan lehet számolni decimális karakterekkel, és az eredmény is decimális lesz. (☛ Az ASCII-kód és a PC-család bővített ASCII-karakterkészlete részletesen megtalálható a *C* függelékben.)

Hogyan címezi a 8088 a tárat

Mivel a 8088 16 bites mikroprocesszor, nem tud közvetlenül 16 bitesnél nagyobb számokkal dolgozni. A 16 biten ábrázolható legnagyobb szám decimálisan 65 535 vagy 64K. Elvileg ez azt jelentené, hogy a 8088 legfeljebb 64 kbájt-os tárat tud elérni. Amint azonban az előző fejezetekben már láttuk, a valóságban ennél sokkal többet is el tud érni: pontosan 1024K-t. Ez a 8088 20 bites címzési sémájának köszönhető, ez bővíti ki a megcímezhető címtartományt 2^{16}-ról (65535-ről) 2^{20}-ra (1048576-ra). A 16 bites feldolgozóképesség mégiscsak korlátozza a 8088-at. A 20 bites cím csak olyan címzési móddal állítható elő, amelyik illeszkedik a 16 bites formához.

A tár kibővítése szegmentált címzéssel

A 8088 processzor tetszőleges számú *szegmensre* osztja a címezhető tárat. Ezek mindegyike legföljebb 64K. Minden szegmens 16-tal osztható címen kezdődik, ezt nevezzük *szegmenscímnek* vagy *szegmensbekezdésnek* (paragrafusnak). Az egyes bájtok vagy szavak eléréséhez szükség van még egy címre, az ún. *eltolási* (vagy offset) *címre*. Ez a szegmenscímmel kijelölt 64K-s szegmensen belül pontosan a kívánt bájtra mutat. Mivel az eltolási címet mindig a szegmens kezdetéhez viszonyítjuk, nevezik *relatív címnek* vagy *relatív eltolásnak* is.



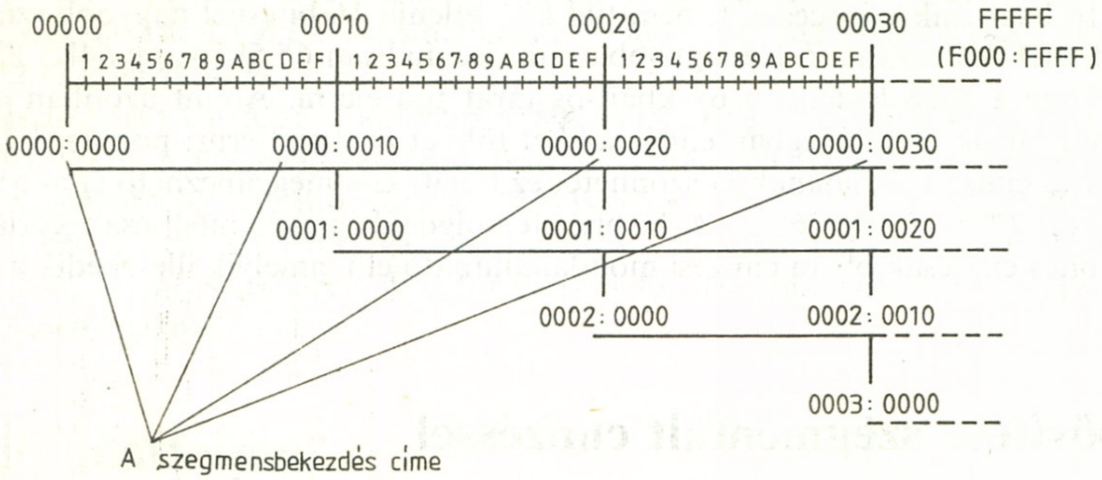
2.1. ábra. A CPU 20 bites fizikai címmé állítja össze a 64K-s tárblokk elejére mutató szegmensbekezdést és a szegmensen belül egy adott bájtra mutató relatív címet

A cím használatakor összeadunk egy 16 bites szegmensbekezdést és egy 16 bites relatív címet. A szegmensbekezdést úgy tekintjük, mintha négy bittel balra lenne léptetve. Ha ezt hozzáadjuk a relatív eltoláshoz, egy teljes 20 bites címet kapunk: ez látható a 2.1. ábrán. A két 16 bites címet együttesen *szegmentált címnek* nevezzük. A megszakításokkal kapcsolatban nevezik még vektornak is. (☛ A megszakítási vektorokról részletesebben a fejezet végén lesz szó).

A szegmensek kezdetét mindig 5 hexadecimális számjeggyel írjuk. Utolsó számjegyük mindig 0: pl. FFE40 vagy B8120. A nulla onnan származik, hogy az eredeti 16 bites, 4

hexadecimális számjegyű értéket 16-tal megszoroztuk. (Hasonló jelenség lép fel a tizes számrendszerben, ha egy decimális értéket a számrendszer alapszámával (10-zel) szorzunk meg: pl. 23 × 10 = 230). Éppen a szegmentált cím szegmensbekezdésének néggyel balra léptetése (vagy az ezzel egyenértékű 16-tal való szorzása) okozza azt, hogy a szegmensek mindig 16-tal osztható címen kezdődnek, és a relatív címre is szükség van a szegmensen belüli pontos cím megadásához. A relatív címet 4 hexadecimális számjeggyel írjuk le. Ha ezt a két számot összeadjuk, 5 hexadecimális számjegyből álló számot kapunk, ez felel meg a 20 bites címnek. Legyen pl. egy szegmensbekezdés címe 1234. Ha ezt 16-tal megszorozzuk, 12340-et kapunk. Ha ehhez hozzáadjuk a keresett bájt relatív címét, mondjuk 4321-et, az 5 számjegyű „hexa" eredményt a következőképpen kapjuk:

12 340	a szegmenscím hexadecimálisan, 4 bittel elléptetve,
+ 4 321	a relatív cím hexadecimálisan,
16 661	a 20 bites szegmentált cím hexadecimálisan.



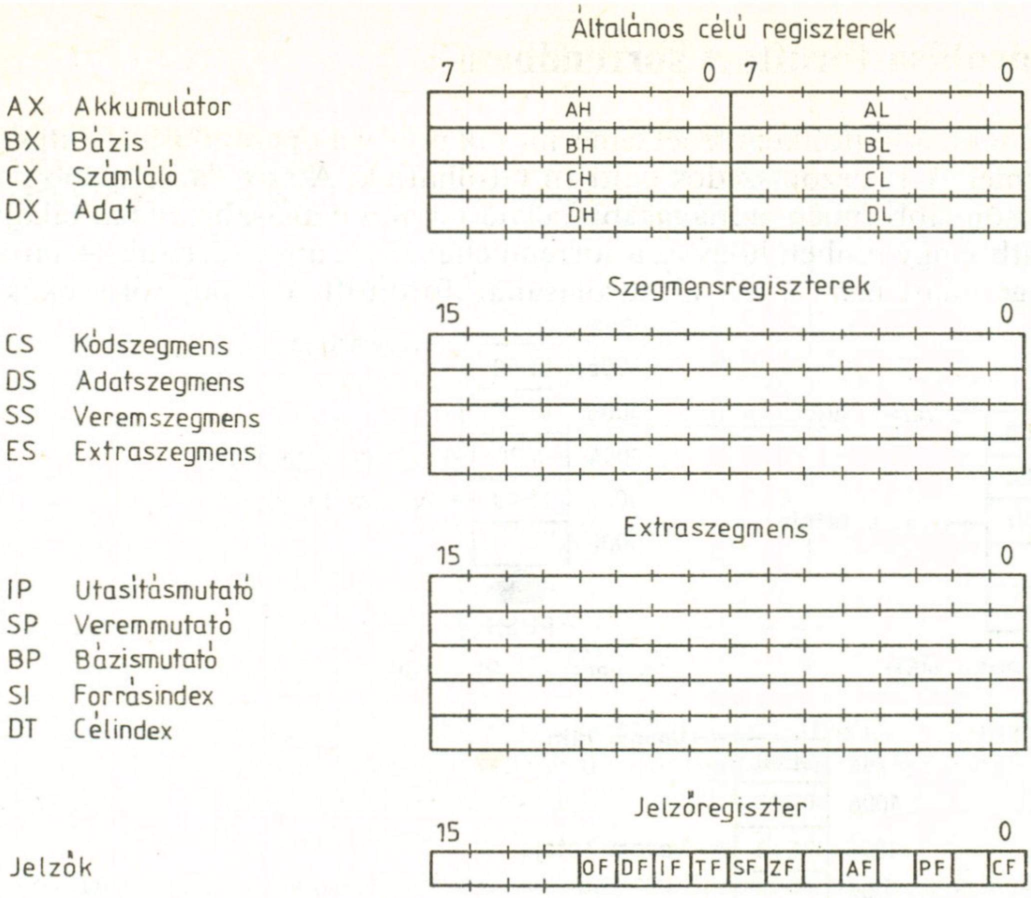
2.2. ábra. A relatív cím mindig a szegmensbekezdéshez viszonyítva értendő, ezért ugyanahhoz a tárcímhez többféle eltérő alakú szegmentált cím tartozhat

Ha a 20 bites címet szegmenscímre és relatív címre bontva írjuk le, akkor a 0000 : 0000 jelölést használjuk – itt bal oldalon áll a szegmenscím, jobb oldalon a relatív cím. Az FFE6E 20 bites címet pl. így írhatjuk fel FFE4 : 002E. Mint a 2.2. ábrán látható, ugyanazt a 20 bites címet többféle módon is felírhatjuk a fenti jelöléssel, attól függően, hogy melyik szegmensbekezdést választjuk.

A 8088 14 regisztere

A 8088-at úgy tervezték, hogy egyidőben tudjon utasításokat, aritmetikai és logikai műveleteket végezni, ill. a tárból utasításokat olvasni és oda adatokat írni. Ehhez 16 bites regisztereket használ.

Összesen 14 regisztere van, a legtöbbnek különleges a feladata, de van négy *általános célú* regisztere is. Ezek tárolják a részeredményeket, az aritmetikai és a logikai műveletek operandusát. Négy *szegmensregiszter* szolgál bizonyos társzegmensek kezdőcímének tárolására. Öt *mutató-* és *indexregiszter* pedig a relatív címeket tárolja. Ezek a szegmenscímmel együtt adják meg az adat pontos tárcímét. Végül van egy *feltételregisztere* is, amelynek 9 db egybites adata feljegyzi a 8088 státuszinformációt, és vezérli a 8088 működését. (☛ L. a 2.3. ábrát.)



2.3. ábra. A 8088 regiszterei és jelzői

Az általános célú regiszterek

Az adatok feldolgozása közben a mikroprocesszor idejének jó részét azzal tölti, hogy adatokat cserél a tárral. Az elérési idő jelentősen lecsökkenthető, ha a gyakran használt operandusokat és eredményeket a 8088-ban tároljuk. Ezt a célt szolgálja a 4 db 16 bites regiszter, vagyis az általános célú regiszterek vagy adatregiszterek.

Az általános célú regiszterek neve *AX, BX, CX* és *DX*. Mindegyik felosztható és megcímezhető, mint 2 db 8 bites fél regiszter. A magasabb helyiértékű fél regiszterek neve *AH, BH, CH* és *DH*; az alacsonyabb helyiértékűeké *AL, BL, CL* és *DL*. A teljes és a fél regiszterek használata az igényeknek megfelelően szabadon keverhető.

Az általános célú regisztereket főleg aritmetikai műveletek átmeneti munkaterületeként használjuk. Az összeadás és a kivonás a tárrekeszek között is elvégezhető, de gyorsabb a regiszterekkel.

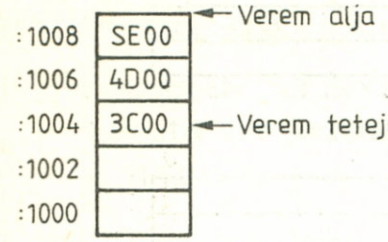
Bár ezek a regiszterek valóban általános célúak, mindegyiknek van valamilyen különleges feladata is.

- Az *AX*-regiszter akkumulátor, az aritmetikai műveleteknél van kitüntetett szerepe.
- A *BX*-regiszter (bázisregiszter) gyakran egy konverziós táblázat kezdetére mutat. Tartalmazhatja egy szegmentált cím relatív részét is.
- A *CX*-regiszter (számláló, count) ciklusok és adatmozgató utasítások számlálójaként használható. A BASIC **LOOP** utasítása pl. *CX*-ben tárolja a ciklus ismétlésének számát. A többi regiszter nem alkalmas erre a feladatra.
- a *DX*-regiszterben 16 bites adatot lehet tárolni általános célokra.

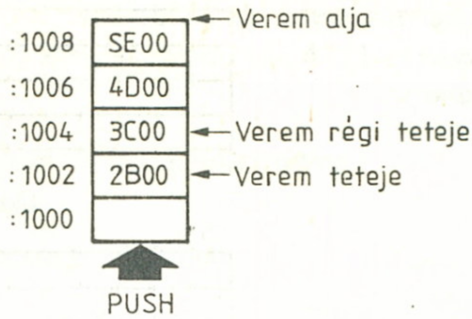
Bár az általános célú regisztereket adatok és operandusok átmeneti tárolására, vagy az előbb említett célra használjuk, de jellegükből adódóan más célokra is felhasználhatók. Bármelyiküket használhatjuk paraméterként átadott adatok relatív címének tárolására.

Szavak tárolása fordított sorrendben

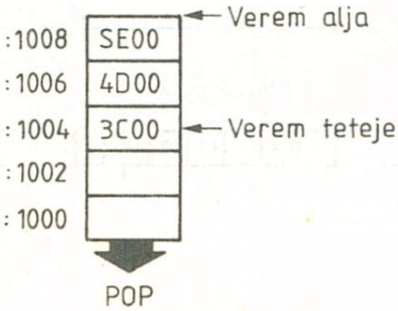
Bár a PC tárát 8 bites bájtonként lehet címezni, sok művelet operandusa 16 bites. A tárban a 16 bitet bármelyik két szomszédos bájtban tárolhatjuk. A szó alacsonyabb helyiértékű bájtját az alacsonyabb címen, a magasabb helyiértékűt a magasabb címen találjuk. Bizonyos nézőpontból úgy tűnhet, hogy ez a sorrend eltér attól, amit várnánk. A tárolási séma fordított jellege miatt néha a szavak tárolásának fordított sorrendjéről szokás beszélni.



A verem a PUSH előtt



A verem a PUSH után



A verem a POP után

Ha szavakkal és bájtokkal egyaránt dolgozunk, akkor vigyázni kell, nehogy belezavarodjunk a sorrendbe. A zavar forrása főleg az adatok írásmódjából ered. Egy szó értékét hexadecimálisan pl. így írhatjuk le: ABCD. A helyiérték azonos a decimális számoknál megszokottal: elől áll a legmagasabb helyiértékű számjegy. Ha azonban a tárolás sorrendjében írjuk le a szót, akkor az alacsonyabb című rekesz tartalmát írjuk előre. A tárban tehát az ABCD számot CDAB sorrendben tároljuk, a bájtok felcserélődnek.

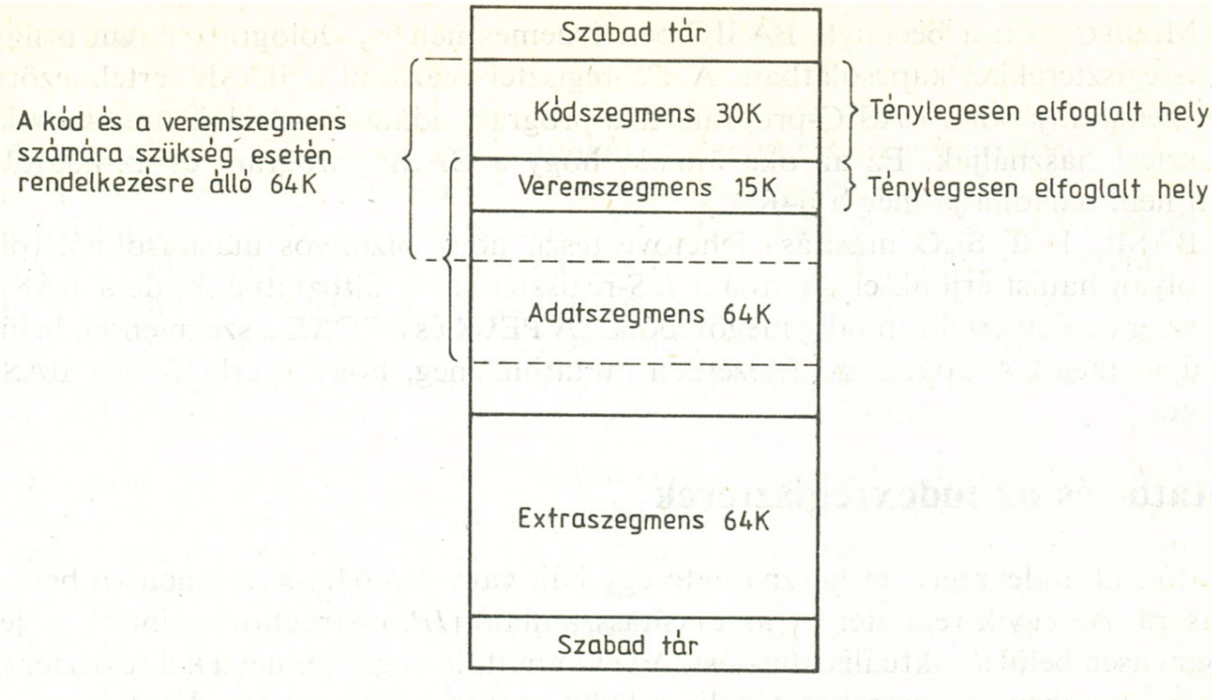
A szegmensregiszterek

Mint korábban már megbeszéltük, egy tárrekesz teljes címe egy 64K-s szegmens címéből és a szegmensen belüli relatív címből tevődik össze. A *CS-*, *DS-*, *SS-* és *ES*-regiszterek tartalmazhatják négy társzegmens címét. A hamarosan ismertetendő 5 eltolási regiszter tárolja a 64K-s szegmensen belüli relatív címet.

A négy regiszterből háromnak különleges feladata van:

- A *CS*-regiszter azonosítja a kódszegmenst; ez tartalmazza a végrehajtandó programot.
- A *DS*-regiszter azonosítja az adatszegmenst; ebben vannak a szükséges adatok.
- Az *SS*-szegmens azonosítja a veremszegmenst (stack); itt tartjuk nyilván a futó program által használt paramétereket és címeket. (☛ A következő oldalakon további adatokat találni a veremről.)

A negyedik szegmensregiszter (az *ES*) az ún. extraszegmensre mutat. Ennek feladata rendszerint az adatszegmens kiegészítése, hogy így 64K-nál nagyobb területeken tárolhassunk adatokat.



2.4. ábra. A szegmensek alkothatnak külön 64K-s területet, vagy átfedhetik egymást. A verem, az adat- vagy az extraszegmens kezdetét úgy állíthatjuk be, hogy a megfelelő regiszterbe – SS, DS, ES – a megfelelő szegmensbekezdés címét töltjük

Megszokott, hogy a négy szegmens átfedi egymást, esetleg azonosak is. Az is gyakori, hogy egy 64K-s szegmensnek csak egy részét használjuk fel rendeltetésének megfelelően. Előfordulhat, hogy egy programnak csak 16K-ra van szüksége egy 64K-s szegmensből. A 2.4. ábrán levő példán egy konkrét tárkiosztás látható.

Minden tárra hivatkozó 8088-as utasítás rejtett módon tartalmazza a megfelelő szegmensregiszter használatát. A **MOV**-utasítás a *DS*-regisztert használja, mivel adatokat mozgat. A **JMP**-utasítás a program végrehajtásának sorrendjét befolyásolja, ezért a *CS*-regisztert használja. A legtöbb esetben felülbírálhatjuk az eredeti szegmensregisztert, ha ez valamilyen okból szükséges. Az assembly nyelven ezt ún. szegmensregisztert felülbíráló prefixummal érhetjük el.

A szegmensregiszter fogalmának és használatának megértése bepillantást nyújt a 16 bites rendszer tárkezelésének gyakorlati korlátaiba. Ha programunk vagy adatainak mérete a 64K-t meghaladja, akkor a most elsajátított ismeretek birtokában meg tudjuk változtatni a szegmensregisztereket. Íme néhány tanács.

Ha nem változtatjuk meg a *CS*-regiszter tartalmát, akkor a program maximális mérete a relatív cím által megszabott 64K. A 8088-at úgy tervezték, hogy tartsa kézben a program irányítását, ezért ha több hely kell a programnak, nem egyszerű feladat a *CS*-regiszter tartalmának közvetlen megváltoztatása. Bizonyos 8088-as utasítások segítségével – pl. távolra ugrással vagy távoli szubrutin hívással – lehet közvetett módon módosítani a *CS* tartalmát. Ezzel a módszerrel biztosítja sok programozási nyelv, hogy a program tetszőlegesen nagy lehessen. (Az értelmezős BASIC és a Microsoft C 1-es változata nem engedi meg az ilyen növekedést, a Pascal és a C 2-es változata viszont lehetővé teszi.)

Viszonylag egyszerű a *DS*-regiszter megváltoztatása vagy az *ES*-regiszter felhasználása 64K-nál több adat kezelésére. Annak ellenére, hogy ez a lehetőség elméletileg korlátlan méretű adatmező létrehozására ad módot, a legtöbb programozási nyelv tervezése legföljebb 64K-s adatmezőt enged meg. Az esetek legnagyobb részében ez nem jelent problémát, mivel a legtöbb program bőven beéri a 64K-s határral. Bár a legfejlettebb programok valóban jól kihasználják a nagy tárterületeket, kevés programnak van szüksége 64K-t megközelítő adatmennyiségre, és még kevesebb tudja valóban fel is használni.

🕮 MEGJEGYZÉS: a beépített BASIC-ben érdemes néhány dologra felhívni a figyelmet a szegmensregiszterekkel kapcsolatban. A *CS*-regiszter igazából a BASIC-értelmezőre mutat. A 8088 szempontjából a BASIC-program és a program adatai is adatnak számítanak, ezért a *DS*-regisztert használják. Ez az oka annak, hogy a BASIC-program és az adatok mérete összesen nem haladhatja meg a 64K-t.

A BASIC **DEF SEG** utasítása lehetővé teszi, hogy bizonyos utasításoknál (pl. **PEEK**, **POKE**) olyan hatást érjünk el, mintha a *DS*-regisztert megváltoztatnánk, de a BASIC-ben a *DS*-regiszter eredeti értéke mindig megőrződik. (A **PEEK** és a **POKE** a szegmensen belüli relatív címet adják meg.) A következő fejezetben mutatjuk meg, hogyan érhető el a BASIC igazi *DS*-értéke.

A mutató- és az indexregiszterek

Öt mutató-, ill. indexregiszter használható egy bájt vagy szó 64K-s szegmensen belüli pontos megadására. Az egyik regiszter – az utasításszámláló (*IP*, instruction pointer) – jelöli ki a kódszegmensen belül az aktuális utasítást. A veremmutató-regiszter nevű két regiszter szorosan kötődik a veremhez. A veremben tárolja a 8088 azokat a címeket és adatokat, amelyekre a későbbiekben szüksége lesz. (☛ A veremről bővebben a következő oldalakon.) A fennmaradó két regiszter (indexregiszter) arra való, hogy az adatszegmensben kijelöljék az aktuális operandust.

Az utasításszámláló (nevezik néha programszámlálónak is) az aktuális kódszegmensen belül kijelöli a végrehajtott utasítást. A *CS*-regiszterrel együtt határozzák meg a következő végrehajtandó utasítás pontos helyét.

A programok közvetlenül nem tudják megváltoztatni az *IP* tartalmát, de egy sor utasítás közvetve megváltoztatja, ill. a verembe menti, vagy onnan visszaállítja azt. Ilyenek pl. a **JMP**- és a **CALL**-utasítások.

A veremmutató-regiszterek (stack pointer, *SP*), ill. bázismutató (base pointer, *BP*) – vermen belüli relatív címet tartalmaznak. Az *SP* a verem pillanatnyi tetejének címét adja meg, és az *IP*-vel analóg. A *BP* segítségével „pillanatfelvételt" készíthetünk a verem tetejének helyzetéről, hogy a későbbiekben pontosan tudjuk, hol is vannak bizonyos adatok a veremben. A *BP* különösen fontos az assembly nyelvű illesztő (interface) szubrutinok szempontjából. Használatával gyakran fogunk találkozni az assembly nyelvű példákban a 8–20. fejezetig.

Az indexregisztereket – nevük forrásindex (*SI*, source index) és célindex (*DI*, destination index) – általában egy másik regiszterrel együtt használjuk. A másik regiszter lehet az *AX*, a *BX*, a *CX* vagy a *DX*, ill. egy utasítás relatív címe. Az utóbbiak tartalmazzák egy adatmező relatív címét az adatszegmensen belül. Az *SI*- és a *DI*-regiszter az adatmezőn belüli relatív címet tartalmazzák. Legtöbbször arra használjuk őket, hogy hosszú karakterláncokat vigyünk át a tár egyik helyéről a másikra. Az *SI*- és *DI*-regisztereket használó karakterlánc műveletek bájtonként vagy szavanként dolgoznak. Az *SI* és a *DI* tartalma általában automatikusan eggyel megnő minden egyes átvitel után, ezért nincs szükség arra, hogy tartalmukat egyesével növeljük, ha a következő bájtra akarunk áttérni.

A feltételregiszter

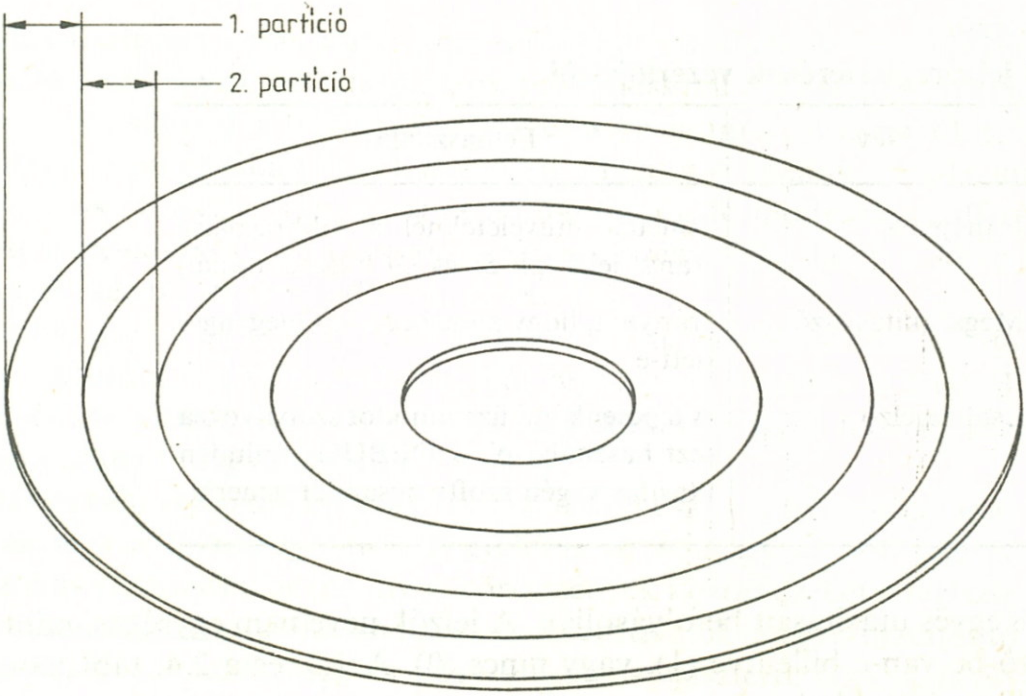
A 14. (és egyben utolsó) regiszter a feltételregiszter. Ez valójában egyedi vezérlőbitek, ún. jelzők (flagek) együttese. A jelzők regiszter formájában állnak rendelkezésre, így mód van rá, hogy adategyüttesként mentsük el, állítsuk vissza, vagy közönséges adatként vizsgáljuk meg. Általában mégis egyesével vizsgáljuk és állítjuk a jelzőket – nem pedig együttesként.

A verem

A verem nem a 8088 processzor része. A verem használatával a program adatokat tárolhat, és nyomon követheti az éppen folyó munkát. A verem segítségével nyilvántarthatjuk azt, honnan hívtak meg egy szubrutint, és milyen paramétereket adtak át neki. A verem átmeneti munkaterületnek is használható, bár ez kevésbé lényeges és kevésbé általános.

Azt mondjuk, hogy az új adatot a verembe tesszük, a régi adatot a veremből kiveszszük. A verem működési elve: elsőnek be, utolsónak ki. Ennek közhasznú angol rövidítése LIFO (last-in-first-out). Ez azt jelenti, hogy amikor a vermet visszatérési címek tárolására használjuk, a legutolsó hívás helyére tér vissza a program először. A verem tehát biztosítja a programok, szubrutinok, megszakításkezelők szabályos működését.

A vermet a teteje felől (a legmagasabb címtől) az alja felé (a legalacsonyabb címig) töltjük fel. Ha tehát adatot teszünk a verembe, akkor az a verem teteje alatti címekre kerül. A verem lefelé növekszik, ha tehát egyre több adatot teszünk a verembe, a verem teteje egyre alacsonyabb címekre kerül, miközben az *SP* tartalma csökken. Ez azzal az előnnyel jár, hogy a verem korábbi tartalmára relatív címekkel (pozitív számokkal) hivatkozhatunk. Mindezt szem előtt kell tartani a verem használatakor, mert gyakran előfordul az assembly nyelvű szubrutinok alkalmazásakor.



Tetszőleges program tetszőleges része új vermet hozhat létre, de ez nem ajánlott. Általában egy futó program létrehoz egy vermet, és ezt használja egész futása alatt a program, annak szubrutinjai, valamint a program futása közben meghívott DOS- és BIOS-szolgáltatások. Ha éppen nem programot futtat, akkor a DOS a saját vermét használja.

Nincs szabály annak meghatározására, hogy mekkora legyen a verem. Ezenfelül a 8088 tervezése sem ad módot annak érzékelésére, hogy mikor fogy el a verem. Sokszor okoz gondot a programozónak, hogy mennyi helyet tartson fönn a veremnek. A programozási nyelveknél (pl. a BASIC-értelmezőnél) szokás 512 bájtos vermet lefoglalni; kivéve, ha más méretet kér a felhasználó. A kivételes esetektől eltekintve ez a méret bőségesen elég.

A 16 bites feltételregiszterben 9 db egybites feltételjelző és 9 db kihasználatlan bit van. A jelzőket két csoportra oszthatjuk. Van 6 státuszjelző, ezek a processzor állapotát mutatják (rendszerint az összehasonlítási és az aritmetikai műveletek eredményét tükrözik), és három kontrolljel-

2.3. táblázat

A 8088 jelzőregiszterének státuszjelzői

Kód	Név	Felhasználás
CF	Átviteljező	Aritmetikai átvitelbit
OF	Túlcsordulásjelző	Aritmetikai túlcsordulása
ZF	Zérusjelző	A nulla eredmény vagy egyenlő mennyiségek összehasonlítása
SF	Előjeljelző	Negatív eredmény (összehasonlítás)
PF	Paritásjelző	Páros számú 1-es bit
AF	Kisegítőátvitel-jelző	Az igazítás szükségességét jelzi a binárisan kódolt decimális (BCD) aritmetikai műveletekben

2.4. táblázat

A 8088 jelzőregiszterének vezérlőjelzői

Kód	Név	Felhasználás
DF	Irányjelző	Ismételt műveleteknél a jobbra/balra irányt jelzi (pl. *SI* és *DI* használatánál)
IF	Megszakításjelző	Irányítja, hogy a megszakítás megengedett-e
TF	Csapdajelző	A lépésenkénti üzemmódot szabályozza (ezt használja pl. a DEBUG): minden utasítás végén szoftvercsapdát generál

ző, amelyek a 8088 egyes utasításait befolyásolják. A jelzők neve nem egységes, mint ahogy az sem, hogy egy jelző be van-e billentve (1), vagy nincs (0). A 2.3. és a 2.4. táblázaton látható elnevezések a legelfogadottabbak.

Tárcímzés regiszterekkel

Már láttuk, hogy a tárcím mindig egy szegmensbekezdés és egy relatív cím összegeként adódik. A cím szegmens része mindig a négy szegmensregiszter egyikétől származik. A relatív rész a következők kombinációjából jön létre:

- egy relatív címből, amelyik magában az utasításban található,
- egy regiszterből: ez rendszerint az *AX*, a *BX*, a *CX* vagy a *DX*, ill.
- az *SI*- vagy a *DI*-indexregiszterből.

Fontos, hogy tisztában legyünk a sokféle lehetőséggel, akár programot írunk, akár olvasunk. Nem minden utasításnál használható minden lehetséges címzési mód. Pl. egy kombináció megengedett, ha az assembler vagy a DEBUG A-parancsa elfogadja.

Tisztában kell lenni a tárcímzésnél és a regisztereknél használt jelölő konvenciókkal is. Szögletes zárójelekkel jelezzük, hogy a zárójelben álló kifejezést relatív címként kell értelmezni. Ez a címzés kulcsfontosságú része. Zárójelek nélkül minden utasításnál a regiszter tartalmát kell használni. Íme néhány példa.

ADD AX, BX	A *BX* tartalmát az *AX* tartalmához adja; tárcímzés nincs
ADD AX, [BX]	Indirekt címzés; a tár egyik rekeszének tartalmát az *AX*-hez adja. A *BX* a cím relatív részét tartalmazza.
ADD [BX], AX	Az *AX*-ben tárolt értéket a tár egyik szavához adja. A *BX* tartalmazza annak a szónak a relatív címét, amihez az *AX* tartalmát hozzáadjuk
ADD AX, 123	Közvetlen címzés: 123-at ad az *AX* tartalmához
ADD AX, [123]	A 123-as relatív címen levő szót az *AX*-hez adja
ADD AX, [BX+SI+123]	Indexelt indirekt címzés: az *AX*-hez adja annak a szónak a tartalmát, amelyiknek a címét úgy kapjuk, hogy a *BX* és az *SI* összegéhez 123-at adunk hozzá.

A regiszterek használatának szabályai

Nagyon fontos tisztában lenni azzal, hogy különböző szabályok vonatkoznak a regiszterek használatára, és e szabályok betartása alapvető fontosságú az assembly szubrutinok írásánál. A szabályok és konvenciók a körülményektől és a programozási nyelvtől is függenek, sajnos nem lehet mindig pontos szabályokat kimondani. Mégis van néhány szabály, amelyik a legtöbb esetre érvényes. (☛ További irányelvek és mintául szolgáló modellek találhatók a 8. és a 20. fejezet között a példákban.) Mindenesetre tartsuk szem előtt, hogy a következő szabályok általában ugyan igazak, de nem abszolút érvényűek.

Az assembly nyelvű illesztő szubrutinban háromféle regiszterhasználat van: egyes regiszterek tartalma szabadon megváltoztatható; más regiszterek tartalma is megváltoztatható, de a szubrutin végén az eredeti tartalmát vissza kell állítani; végül néhány regiszter tartalmát egyáltalán nem szabad megváltoztatni.

Az általános célú regisztereket (*AX*-től *DX*-ig) rendszerint nyugodtan meg lehet változtatni, ez nem zavarja a hívóprogramot. Figyeljünk arra, hogy az *AX* regiszterben általában eredményt szokás visszaadni, de bizonyos körülmények között ezeket a regisztereket paraméterek átadására használják, és felhasználásuk után tartalmuk megváltoztatható.

Külön szabályok vonatkoznak a négy szegmensregiszterre (*DS, CS, SS* és *ES*). A *CS*-regisztert soha nem szabad közvetlenül megváltoztatni, de távoli és közeli szubrutinhívásokkal közvetett módon megváltoztatható. A *DS*-regiszter tartalma is megváltoztatható, de általában utána helyre kell állítani az eredeti tartalmát. Az *SS*-regiszter tartalmának eredeti értékét mindig meg kell őrizni, ha megváltoztatjuk. Általában a szubrutinok azt a vermet szokták használni, amire az *SS* mutat, de ha saját vermet hoznak létre, akkor a szubrutin végén vissza kell állítani az *SS* eredeti értékét. Gondoljunk arra, hogy az *SS* értékének megváltoztatása befolyásolhatja a paraméterek elérésére használt bázismutató (*BP*) használatát. Az *ES*-regisztert általában tetszés szerint meg lehet változtatni.

Az utasításszámlálót (*IP, PC*) nem szabad direkt módon megváltoztatni; mert akárcsak a *CS*-regiszternél, az indirekt változtatások automatikusak és pontosak.

A veremmutató megváltoztatható, de általában az *SP* közvetlenül változik, annak következményeként, hogy a verembe beteszünk, vagy abból kiveszünk valamit. A veremmutató visszaállítása (alapállapotba hozása) a szubrutinok használatának egyik alapvető konvenciója; az erre vonatkozó szabályok változók. (☛ L. a 8. és a 20. fejezetben közölt példákat.)

A bázismutatót (*BP*) általában meg kell változtatni ahhoz, hogy a paraméterekhez hozzáférjünk; gyakran helyre is kell állítani.

Az indexregisztereket (*SI* és *DI*) szükség szerint szabadon lehet változtatni.

A feltételregiszterben az állapotjelzőket nyugodtan megváltoztathatjuk. Gondoljunk rá, hogy az állapotjelzők néha eredményt közölnek, beállításuk tehát lényeges lehet. Leggyakrabban a *CF*-t és a *ZF*-t használják erre a célra. Ami a kontrolljelzőket illeti, a megszakításjelzőt (interrupt flag, *IF*) célszerű beállítva hagyni (megszakítás engedélyezve), az irányjelzőt (direction flag, *DF*) is tanácsos beállítva hagyni, a lépésenkénti programvégrehajtás jelzőjének (trap flag, *TF*) beállítása viszont kész öngyilkosság.

Hogyan használja a 8088 a portokat

A 8088 a számítógép sok részével kiviteli és beviteli portokon keresztül kommunikál, ill. azokon keresztül vezérli őket. A portok olyan kapuk, amelyeken az információ áthalad, mialatt egy bemeneti/kimeneti (input/output, I/O) berendezés felé halad, vagy onnan indul el. Ilyen berendezés lehet pl. a billentyűzet vagy a nyomtató. Az 1. fejezetben leírt legtöbb kiszolgáló IC használja a portokat; sőt a legtöbb IC több portot is használ különböző célokra.

A portokat egy 16 bites portszámmal azonosítjuk, ennek értéke 0 és 65 535 közé esik. A CPU úgy küld adatot vagy vezérlőinformációt egy bizonyos portnak, hogy kiküldi a portnak az azonosító számát. A port adat- vagy állapotinformáció visszaküldésével válaszol.

Akárcsak a tár elérésénél, a CPU itt is a cím- és az adatbuszt használja adathordozónak a porttal való kommunikáció során. A port kiválasztása úgy kezdődik, hogy a vezérlőbuszon a CPU egy jelet küld ki, ez az összes I/O berendezéssel tudatja, hogy a címbuszon következő adat egy port címe. Ezután kiküldi a port címét. A megfelelő című port válaszol.

A portszám egy olyan tárrekeszt címez meg, amely az I/O berendezés része, nem a központi táré. Különleges I/O utasítások jelzik a porthoz férést, és küldik az adatot a berendezés és a CPU között. Néhány I/O berendezés (pl. a videokontroller) az I/O portjain kívül a központi tár címeit is felhasználja. Ezekről a CPU azt hiszi, hogy a RAM részét képezik. Ezt tárra leképezett I/O-nak nevezik. Általában könnyebb a tárra leképezett berendezéseket programozni, mert ezeknél használhatók a rugalmas tárkezelő utasítások is, a meglehetősen rugalmatlan és korlátozott I/O utasítások helyett.

📖 MEGJEGYZÉS: A 8088 utasításkészletében az **IN** és az **OUT** utasítással tudjuk az adatokat portra írni és onnan beolvasni. A BASIC-ben található **INP** és **OUT** utasításokkal is lehetséges az, hogy BASIC-szubrutinok segítségével kísérletezzünk az egyes portokkal, és assembly nyelvű programozás nélkül is beépíthessük a portok kezelését programjainkba.

Családon belüli különbségek a portok használatában

Az egyes portok használatát a hardver tervezői határozzák meg. A portok használóinak ismerniük kell a portok számát, használatukat és jelentésüket. Mivel a portok kiosztása némileg eltér a PC-család tagjai között, a 2.5. táblázatban felsoroltuk a szabványos portokat, számaikat és használatukat. Sokszor a felhasználásuk is közös az egész PC-családon belül, azonban figyeljük meg, hogy a PCjr és az AT némileg eltér a portok számozásában.

(☛ A portcímek használata előtt tanácsos elolvasni az IC-k leírását az 1. fejezetben. A 7. fejezetben foglalkozunk a hanggenerálással, és ott példákkal illusztrálják, hogyan lehet a portok segítségével a hangadásra közvetlenül a hardvert programozni.)

2.5. táblázat

A PCjr, a PC/XT és az AT gépekben használt portok és portcímek

Leírás	Intervallum		
	PCjr	PC/XT	AT
DMA-vezérlő (8237A-5)	n/a	000–00F	000–01F
Megszakításvezérlő (8259A)	020–027	020–021	020–03F
Időzítő (8253–5; 8254. 2 az AT-ben)	040–047	040–043	040–05F
PPI (8255A–5; 8255 – 5 a PCjr-ben)	060–067	060–063	n/a
Billentyűzet (8042)	n/a	n/a	060–06F
DMA-oldal regisztere (74LS612)	n/a	080–083	080–09F
NMI (nemmaszkolható megszakítás) maszk regisztere	0A0–0A7	0A	070–07F
2-es megszakításvezérlő (8259A)	n/a	n/a	0A0–0BF
Hanggenerátor (SN76496N)	0C0–0C7	n/a	n/a
DMA-vezérlő 2(8237A-5)	n/a	n/a	0C0–0DF
Matematikai társprocesszor törlése/alapállapotba hozása	n/a	n/a	0F0–0F1
Matematikai társprocesszor	n/a	n/a	0F8–0FF
Botkormány (játék irányításához)	200–207	200–20F	200–207
Bővítő egység	n/a	210–217	n/a
Párhuzamos nyomtató (másodlagos)	n/a	n/a	278–27F
Soros port (elsődleges)	2F8–8FF	3F8–3FF	3F8–3FF
Soros port (másodlagos)	n/a	2F8–2FF	2F8–2FF
Próbakártya	n/a	300–31F	300–31F
Merevlemez	n/a	320–32F	1F0–1F8
Párhuzamos nyomtató (elsődleges)	n/a	378–37F	378–37F
SDLC (másodlagos biszinkron kommunikáció; csak az AT-nél)	n/a	380–38F	380–38F
Biszinkron kommunikáció (elsődleges)	n/a	n/a	3A0–3AF
Monokrom adapter/nyomtató	n/a	3B0–3BF	3B0–3BF
Színes-grafikus adapter	n/a	3D0–3DF	3D0–3DF
Hajlékonylemezes vezérlő	0F0–0FF	3F0–3F7	3F0–3F7

Bizonyos portokra való írással teljesen szét lehet zilálni a gép működését, de egyetlen portolvasás is járhat nem kívánt hatással. Hibás az a feltételezés, hogy egy port kiolvasása nem befolyásolja a számítógép működését, mint ahogy az is, hogy amit nyugodtan megtehetünk az egyik PC-modellen, azt a család minden gépén megtehetjük. A következő program pl. tökéletesen fut a legtöbb PC-modellen, de leblokkolja a PCjr-t.

```
10 FOR I = 50 TO 75
20  IF I = 64 THEN PRINT „Mi történik most?"
30  PRINT I, INP (I)
40 NEXT I
```

A program a BASIC **INP**-parancsával adatokat olvas be egy portról. (Az **INP** jelentése IN portról.) Megpróbál beolvasni az 50 és a 75 közötti portokról; ez látszólag ártalmatlan tevékenység. Mégis, ez a rövid program leblokkolja a PCjr-t, amint a 64-es porthoz ér. A többi PC-n teljesen simán lefut. Az okok a hardver homályos részleteibe vesznek.

Hogyan használja a 8088 a megszakításokat

Valahányszor egy hardverberendezésnek vagy egy programnak a CPU közreműködésére van szüksége, egy *megszakítás*nak nevezett jelet vagy utasítást küld a mikroprocesszornak. A jel vagy utasítás egyben azonosítja a végrehajtandó taskot is. Amikor a mikroprocesszor megkapja a megszakításjelet, általában befejezi minden más tevékenységét, és elindítja a megszakításhoz tartozó megszakításkezelő szubrutint. Miután a megszakításkezelő befejezte feladatát, a gép onnan folytatja tevékenységét, ahol a megszakítás bekövetkezésekor tartott.

A megszakításoknak három fő csoportja van. Az első csoport, amit a számítógép áramkörei generálnak, ha valami történik (pl. egy billentyű lenyomása). Ezeket a megszakításokat a megszakításvezérlő IC (a 8259) kezeli, amely egyben fontosság szerint rendezi is a megszakításokat, mielőtt a CPU felé továbbítaná őket. A második csoportba tartozó megszakításokat a CPU generálja. Ezek a programban fellépő rendkívüli esemény következményei (ilyen pl. a nullával végzett osztás). Végül a harmadik csoport megszakításait a program szándékosan idézi elő, ily módon hív meg egy ROM-ban vagy RAM-ban tárolt távoli szubrutint. Ezek a megszakítások – gyakran nevezik szoftvermegszakításnak – rendszerint a ROM-BIOS és a DOS szolgáltatások részei. (☛ Részletesen lesz róluk szó a 8 – 18. fejezetekben.) Mód van a szoftvermegszakítás szubrutinjainak megváltoztatására, sőt újak írására, ha a helyzet úgy kívánja.

Az előzőeken kívül van egy különleges megszakítástípus is: ezt *nemmaszkolható megszakítás*nak (**NMI**, non-maskable interrupt) nevezzük. Olyankor használják, ha a CPU azonnali intézkedésére van szükség. Gyakran veszélyhelyzetet jelez, pl. áramkimaradást vagy memóriahibát. Az NMI megjelenésekor ez kapja a legnagyobb prioritást, és a CPU a többi megszakítási kérelem előtt szolgálja ki.

Bármilyen módon keletkezik a megszakítási kérelem, a megszakítást kérő forrásnak nem kell ismernie a kért megszakításkezelő címét; elég a megszakítás sorszámát ismernie. Ez a szám a tár legalacsonyabb címein tárolt táblázatra mutat, a táblázat pedig a megszakításkezelő szubrutinok szegmentált címét tartalmazza. A megszakításkezelő szubrutinok címeit *megszakítási vektor*nak nevezzük, a táblázatot pedig megszakítási vektorok táblázatának. A táblázat a BIOS vagy a DOS felügyelete alatt áll. (☛ Részletesebben foglalkozunk ezzel a 3. fejezetben.) Amikor új megszakításkezelő szubrutint írunk, akkor vagy egy már létező számát és vektorát használjuk fel, vagy új számot jelölünk ki hozzá.

A megszakítások automatikusan a verembe mentik a kódszegmensregiszternek (*CS*) és az utasításszámlálónak (*IP*) az értékét, ez biztosítja, hogy a gép ott tudja folytatni a futást, ahol a megszakítás bekövetkezett. Ezenkívül a megszakítási folyamat menti a feltételregisztert, és nullára állítja a megszakításjelzőt (*IF*), ezzel ideiglenesen letiltja a további megszakításokat. Általában a megszakításkezelő szubrutin amint lehet, újra engedélyezi a megszakításokat. Ez legtöbbször az első néhány utasítás után megtörténik. Van erre egy különleges – a megszakítási szubrutinból visszatérő – **IRET** nevű utasítás is. Megfelel a szubrutinhívásnál használt **RET**-utasításnak, helyreállítja a jelzőket, a *CS*-t és az *IP*-t.

Szokványos dolog assembly szubrutinokat hozzászerkeszteni a programokhoz, ill. programozási nyelvekhez. Ily módon hozzáférünk a BIOS és a DOS szolgáltatásaihoz, vagy más módon növeljük segítségével a program teljesítményét. Ilyen illesztő szubrutinoknál (különösen amelyek a DOS-t vagy a BIOS-t hívják) elengedhetetlen, hogy assembly nyelven lehessen programozni. Legtöbbször megfelel, ha ezek az illesztő szubrutinok mindössze szubrutinhívásokból és **RET**-utasításokból állnak, esetleg megszakításkérésből az **INT**-utasítás segítségével. Csak a legkifinomultabb assembly nyelvű programozásnál van szükség megszakításkezelők írására és az **IRET**-utasítás használatára.

3 A ROM-szoftver

Bevezetés • 42

Az indító ROM • 42

A ROM-BIOS • 43

Megszakítási vektorok • 44
A legfontosabb címek a tár alján • 49
A ROM-változat és a gép azonosítója • 55

A ROM-BASIC • 58

A ROM bővítései • 58

Cserélhető szoftver

Megjegyzések

Bevezetés

A számítógép működéshez szoftverre van szükség. A számítógép elindítása és működésben tartása sokkal könnyebb, ha a szoftver egy része mindig a számítógépben van. Pontosan ezért vannak ROM programok. A ROM (a read-only memory, csak olvasható memória) olyan memória, amelynek a tartalma kitörölhetetlenül rögzítve van a PC áramköreiben. Ezeknek a tartalma nem károsodhat, nem törlődhet, és nem veszhet el.

A PC-kben tekintélyes mennyiségű ROM van. Ezekben vannak a PC elindításához, működésben tartásához és a periferiális berendezések működtetéséhez szükséges programok. Azért előnyös a programok nagy részét ROM-ban tárolni, mert így a számítógépbe beépítve mindig jelen vannak, nem szükséges őket lemezről a tárba tölteni, mint pl. a DOS-t. Jellegüknél fogva a ROM-programokra (mint alapkövekre) épül a legtöbb program pl. a DOS is!

A PC-család ROM-jaiban négyféle program van: az indító programok – ezek végzik a gép indítását – ; a ROM-BIOS – a betűszó a BASIC INPUT/OUTPUT SYSTEM rövidítése (alapvető beviteli/kiviteli rendszer) –, a gép működéséhez szükséges gépi kódú kiszolgáló szubrutinok gyűjteménye; a ROM-BASIC – a BASIC-értelmező magja – és a ROM bővítései, amit akkor tesznek a gépbe, ha új hardverelemmel bővítik azt. A fejezet hátralevő részében ezt a négy fő alkotórészt vizsgáljuk meg.

Az F000 szegmensbekezdésnél kezdődő, legmagasabb című tárblokk van fenntartva a ROM-programok számára. A PC-család különböző modelljei eltérő mértékben veszik igénybe ezt a 64K-s blokkot, attól függően, hogy mennyire bonyolult a ROM-szoftver. Pl. a meglehetősen egyszerű hardver felépítésű eredeti PC-modellben a ROM-programok 40K-t használnak, míg a PCjr-ben és az AT-ben 64K-t, mivel hardverjük jóval bonyolultabb.

Az indító ROM

A ROM-programok első feladata a gép bekapcsolásának felügyelete. A ROM többi részétől eltérően az indító szubrutinoknak kevés köze van a PC-család programozásához, mégis érdemes megismerkedni tevékenységükkel.

Az indító szubrutinok többek között a következőket hajtják végre: lefuttatnak egy gyors megbízhatósági tesztet (a számítógépre és a ROM programokra); inicializálják az IC-ket és a géphez csatlakozó szabványos berendezéseket; kitöltik a megszakítási vektorok táblázatát; ellenőrzik, hogy milyen kiegészítő berendezés kapcsolódik a géphez; és ha van lemez, akkor befejezésül legtöbbször betöltik róla az operációs rendszert.

A megbízhatósági teszt – amely a bekapcsolási önteszt (POST, power on self test) része – első lépése annak a folyamatnak, amellyel meggyőződhetünk arról, hogy a gép rendben van. A POST-tesztek nagy része elég rövid, kivéve a memóriateszteket. Ez utóbbiak bosszantóan sokáig tartanak, ha a gép tára nagy.

Az inicializálási folyamat valamivel bonyolultabb. Az egyik szubrutin beállítja a megszakítási vektorok alapértékét. Az alapértékek vagy a ROM-BIOS szabványos megszakításkezelőire mutatnak, vagy ún. üres szubrutinokra; ezeket később a mi programjaink töltik fel. Egy másik inicializáló szubrutin azt ellenőrzi, hogy milyen berendezések vannak a géphez kapcsolva, aztán a megfelelő bejegyzést elhelyezi a memória alacsony című részében kijelölt helyen. (A fejezet hátralevő részében részletesen megbeszéljük ezt a berendezéslistát.) A berendezés jelenlétét minden modell másképp állapítja meg.

PC A PC-nél a rendszerkártyán elhelyezett kapcsolók beállításából. JR A PCjr-nél (logikai vizsgálattal és teszteléssel) az inicializáló program „rákiált" minden lehetséges berendezésre: „Itt vagy?" – és figyeli a választ. AT Az AT az adatokat egy különleges tárból olvassa ki, amelynek tartalma kikapcsoláskor se vész el. (A tár tartalmát diagnosztikai programokkal lehet beállítani.)

Függetlenül attól, hogy milyen módon szerezték be az adatot, az információ feljegyzése és tárolása minden modellnél azonos módon történik, és a programból megvizsgálható. Az inicializáló szubrutinok figyelnek arra is, hogy van-e új berendezés és ROM-bővítés. Ha találnak ilyet, akkor a vezérlést ideiglenesen a ROM-bővítésnek adják át, hogy azok is inicializálhassák magukat. Ezután az inicializáló szubrutinok végrehajtják a hátralevő indító szubrutinokat. (Erről később részletesebben szólunk.)

A bekapcsolási eljárás utolsó része – a POST-tesztek, az inicializáló eljárás és a ROM-bővítések beépítése után – az ún. *öntöltő* vagy *boot-strap*. Ez egy rövid szubrutin, segítségével egy program tölthető be a lemezről. Lényegét tekintve a ROM öntöltője megkísérli az ún. betöltőrekord beolvasását a lemezről, és ha sikerül, átadja a gép vezérlését a rekordban tárolt programnak. A lemez öntöltő rekordjában tárolt program feladata a lemezes program többi részének betöltése. Ez a program rendszerint a lemezes operációs rendszer (a DOS), de éppúgy lehet egy teljesen önálló, önmagát betöltő program is, pl. a Microsoft Flight Simulator. Ha a ROM-öntöltő nem tudja leolvasni a lemez öntöltő rekordját, akkor egyszerűen aktivizálja a ROM-ban tárolt „kazettás" BASIC-et. (A bővebb PC-család nem IBM gyártmányú tagjainál az utóbbi lépés helyett sikertelen betöltésre utaló üzenet jelenik meg.) Amint a fenti két lehetőség valamelyike befejeződik, a bekapcsolási eljárás lezárul, és készen állunk arra, hogy más programok vegyék át a gép irányítását.

📖 MEGJEGYZÉS: A ROM-bővítések megváltoztathatják vagy megakadályozhatják a betöltő eljárást. JR A fejezet vége felé látni fogjuk, hogy ez leginkább a PCjr cserélhető szoftvereinél vehető észre.

A ROM-BIOS

A ROM-BIOS a ROM-tárnak az a része, amelyet a gép teljes bekapcsolási ideje alatt aktívan használ. A ROM-BIOS biztosítja azokat az alapvető szolgáltatásokat, amelyek nélkülözhetetlenek a gép működéséhez. Döntő részben a BIOS vezérli a számítógép perifériáit, a képernyőt, a billentyűzetet és a lemezmeghajtókat. Legszűkebb értelemben a BIOS a berendezéseket vezérlő programokat jelenti. Ezek a programok alakítják át az olyan egyszerű parancsot: „Olvass valamit a lemezről!" – azokká az elemi lépésekké, amelyek a parancsot valóban végrehajtják, beleértve a hibák érzékelését és javítását is. Legtágabb értelemben a BIOS azokat a szubrutinokat is jelenti, amelyek a számítógép működésének szempontjából alapvető fontosságú információkat adnak (pl. a napi idő nyilvántartását).

Lényegében a BIOS-programok a hardver és a mi programjaink – a DOS-t is ideértve – között helyezkednek el. Végeredményben ez azt jelenti, hogy a BIOS két irányban működik, mint egy kétoldalú folyamat. Az egyik oldalhoz kérelmek érkeznek a programoktól, hogy hajtsa végre valamelyik szabványos I/O feladatot. Ezeket a szolgáltatásokat programjaink úgy veszik igénybe, hogy megadják a megszakítás sorszámát (ez jelzi a megszakítási kérelem tárgyát, pl. a nyomtatást); és egy paramétert (ez jelzi pontosan, mit is kell végrehajtani). A BIOS másik oldala a számítógép hardveregységeivel kommunikál – a képernyővel, a lemezmeghajtókkal stb. – a berendezések működtetéséhez szükséges parancskódokkal. A BIOS-

nak ez az oldala kezeli azokat a hardvermegszakításokat is, amelyekkel a berendezések akarják magukra felhívni a figyelmet. Pl. valahányszor megnyomunk egy billentyűt, megszakítás jön létre, hogy a BIOS értesüljön az eseményről.

A teljes ROM-szoftverekből valószínűleg a BIOS-szolgáltatások a legfontosabbak a programozó számára, ez az oka, hogy 5 fejezetben (8.-tól a 13.-ig) foglalkozunk ezzel. Mivel később még részletezni fogjuk ezt, most kihagyjuk a BIOS-szolgáltatások ismertetését. Inkább arra összpontosítjuk figyelmünket, hogy a BIOS egységes egészként hogyan tartja kézben a számítógép I/O folyamatát.

Megszakítási vektorok

Az IBM PC-családot (mint az Intel 8086-ra alapozott összes számítógépet) igen nagy részben megszakítások vezérlik. Ezeket mind a hardver, mind a szoftver előállíthatja. Ez alól a BIOS kiszolgáló szubrutinjai sem kivételek, mindegyikhez megszakítási sorszám tartozik; ezt kell megadni, ha a szolgáltatást igénybe akarjuk venni.

Amikor megszakítás lép fel, a számítógép vezérlését egy megszakításkezelő szubrutin veszi át. Ez a szubrutin gyakran ROM-ban van (egy BIOS kiszolgáló szubrutint tekinthetünk egyszerűen megszakításkezelőnek is). A megszakításkezelő meghívásakor a szegmentált cím a program futását vezérlő regiszterekbe töltődik: azaz a *CS* (kódszegmens) és az *IP* (utasításszámláló) regiszterekbe. (Ennek a kettősnek szokásos jelölése: *CS:IP*.) A megszakításkezelőket azonosító szegmentált címet szokás megszakítási vektornak nevezni.

A bekapcsolási eljárás során a megszakítási vektorokba betöltődik a ROM-ban tárolt megszakításkezelők címe. Ezek a címek egy RAM-ban található táblázatban vannak. Minden cím két szót foglal el: az első a relatív cím, a második a szegmenscím. A 8088 az egyes címeken belül fordított sorrendben tárolja a bájtokat. (☞ A 2. fejezetben ezt részletesen megmagyaráztuk.) A megszakítási vektor tartalmát egyszerűen megváltoztathatjuk, hogy egy új megszakításkezelőre mutasson.

A PC-család megszakításait hét csoportra oszthatjuk: a mikroprocesszor, a hardver, a szoftver, a DOS, a BASIC, a címzés megszakításokra és az általános célú megszakításokra.

A mikroprocesszor megszakításait gyakran nevezik logikai megszakításnak. Közülük négyet (a 0-ás, 1-es, 3-as, 4-es megszakítást) maga a mikroprocesszor generál, egy továbbit (a 2-es megszakítást, azaz az **NMI-t**) egy külső berendezés által előállított jel aktivizál.

A hardver megszakításai a PC hardverjébe vannak építve. E megszakítások közül 8 vagy a mikroprocesszorba, vagy a rendszer kártyába van bedrótozva, és nem változtatható meg. Az összes hardver megszakítás a 8259A IC felügyelete alatt áll. A fenntartott kódok a 2-es, 8-as, 9-es és a 11–15-ös.

A szoftver megszakításai a PC tervezésének szerves részét képezik és a ROM-BIOS-programoknak is része. A megszakítások által meghívott BIOS-szubrutinokat magukat nem lehet megváltoztatni, de a rájuk mutató vektorok átírhatók úgy, hogy más szubrutinra mutassanak. A fenntartott kódok: az 5-ös, a 16–28-as és a 72-es.

A DOS szerepe

Miután a ROM-öntöltő programja a lemez betöltőrekordjának adta át a vezérlést, a betöltőrekord ellenőrzi, hogy a lemezen van-e a DOS. Azt is megvizsgálja, hogy rajta van-e a lemezen két rejtett programfájl; az IBMBIO.COM és az IBMDOS.COM. Ha megtalálja

őket, akkor a DOS parancsértelmezőjével együtt (neve COMMAND.COM) betölti őket. A betöltési folyamat során a DOS esetleges bővítő része – pl. a beépíthető berendezésvezérlők – is betölthető.

Az IBMBIO.COM fájl a ROM-BIOS kiterjesztését tartalmazza. A bővítés lehet pl. a meglévő ROM-BIOS változtatása vagy kiterjesztése, gyakran a meglévő ROM-BIOS hibáinak javítását tartalmazza: új berendezésekhez szükséges új szubrutinokat, vagy a szabványos BIOS-szubrutinok „testreigazítását". Mivel az IBMBIO.COM szubrutinok lemezen vannak, segítségükkel kényelmesen módosítható a ROM-BIOS. Az új szubrutinon kívül elegendő a megszakítási vektort úgy módosítani, hogy a régi helyett az új szubrutin kezdőcímére mutasson. Ha egy új berendezéssel bővül a számítógép, akkor vezérlő programja elhelyezhető az IBMBIO.COM fájlban, vagy beépíthető berendezésvezérlőként, nincs szükség a ROM cseréjére. (☛ Az *A* függelékben további részletek találhatók a beépíthető berendezésvezérlőkről.)

A ROM-BIOS szubrutinjait úgy képzelhetjük el, mint a legfontosabb és legegyszerűbb I/O műveleteket végrehajtó legalacsonyabb szintű rendszerszoftvert. Az IBMBIO.COM szubrutinjai (melyek a ROM-BIOS bővítései) ugyanezen a szinten helyezkednek el, ugyancsak alapvető feladatokat látnak el. Összehasonlításképpen az IBMDOS.COM szubrutinjai ennél összetettebbek; úgy képzelhetjük el őket, hogy egy szinttel feljebb helyezkednek el, a hierarchia legtetején pedig a programozási nyelvek vannak.

Az IBMDOS.COM fájl tartalmazza a DOS kiszolgáló szubrutinjait. A DOS-szolgáltatásokat (a BIOS-szolgáltatásokhoz hasonlóan) megszakítások segítségével vehetjük igénybe. A DOS-szolgáltatások vektorai is a megszakítási vektorok táblázatában találhatók, a központi tár alacsony című részén. A 33-as (hexadecimális 21) DOS-megszakítás különösen fontos, mert ezen keresztül egy sor másodlagos szubrutin, az ún. DOS-függvények, érhetők el. A DOS-függvények a BIOS-szubrutinoknál összetettebb és hatékonyabb ellenőrzési lehetőséget biztosítanak az I/O műveletek fölött – különösen igaz ez a lemezes műveletekre. A szabványos lemezműveletek – pl. lemez formázása; adatok írása és olvasása; fájl megnyitása, lezárása és törlése; keresés könyvtárakban – kivétel nélkül DOS-függvények. Ezek képezik az alapját egy sor magasabb szintű DOS-parancsnak, pl. a **FORMAT**-nak, **COPY**-nak, **DIR**-nek. Egy program akkor szokta igénybe venni a DOS-szolgáltatásokat, ha szorosabban akarjuk kézben tartani az I/O műveleteket, mint amenynyire a programozási nyelvek lehetővé teszik, de nem akarunk leásni a BIOS szintjére sem. A DOS-szolgáltatások e könyv fontos részét képezik, velük foglalkozunk a 14–18. fejezetig.

Hasznosság szempontjából legfontosabb a 3. fájl, a COMMAND.COM. Ebben találhatók azok a szubrutinok, amelyek a DOS parancsmódjában beírt üzeneteket értelmezik. A COMMAND.COM a beírt üzenetet a parancsok neveivel hasonlítja össze – ezek a nevek egy táblázatban vannak –, így meg tudja különböztetni a COMMAND.COM részét képező belső parancsokat (**RENAME** vagy **ERASE**) a külső parancsoktól, a DOS karbantartó programjaitól (a DEBUG-tól) vagy a felhasználó programjaitól stb. A beírt üzenetre, ha az belső parancs, végrehajtja a megfelelő programot; ha nem, megpróbálja lemezen megkeresni és betölteni a kért programot. Maga a COMMAND.COM fájl és működése nagyon érdekes, megéri a bővebb tanulmányozást (akárcsak a többi DOS-program). A DOS Technical Reference vagy az Inside the IBM PC elolvasása ajánlható azoknak, akik további részletekre kíváncsiak.

A DOS-megszakítások a DOS működése közben mindig igénybe vehetők. Sok program és programozási nyelv használja a DOS-megszakításokat az alapműveletek – főleg a lemezes I/O – kezelésére. A 32-es és a 255-ös közötti kódok vannak erre a célra fenntartva. (Jelenleg a 32-es és a 96-os közöttiek használatosak, a többi tartalék.)

*A BASIC megszakításai*t maga a BASIC osztja ki, és a BASIC futása közben használhatók. Ezek a 128-as és a 240-es közötti kódok lehetnek.

A címzés megszakításai a megszakítási vektorok táblájában találhatók, és szegmentált címet tartalmaznak. Ezekhez a megszakításokhoz nem kapcsolódik tényleges megszakítás vagy megszakításkezelő szubrutin. Közülük három, egy-egy fontos táblázathoz kapcsolódik: a videoinicializáló táblázathoz, a lemez alaptáblázatához és a grafikus karakterek táblázatához. Ezekben a táblázatokban olyan adatok vannak, amelyeket a ROM-BIOS használ fel a bekapcsoláskor és a grafikus karakter előállításakor. A 29–31-es és a 68-as, 73-as kódok vannak erre a célra fenntartva (JR A 68-asat és a 73-asat csak a PCjr használja.)

*Az általános célú megszakítások*at a felhasználói programok hozzák létre ideiglenes jelleggel. Ezek a 96–103-asig terjedő kódok lehetnek.

A megszakítási vektorok a központi tár legalacsonyabb című részén találhatók, az első rekesz a 0-ás megszakítás vektorát tárolja, ill. sorban a többit. Mivel egy megszakítási vektor 2 szót foglal el, egy adott megszakítás címének helyét úgy találjuk meg a tárban, hogy sorszámát 4-gyel megszorozzuk. Pl. a képernyő tartalmát kinyomtató 5-ös megszakítás a 20-as címen található ($5 \times 4 = 20$). Ezt a megszakítási vektort úgy vizsgálhatjuk meg a DEBUG-gal, hogy értékét hexadecimálissá alakítjuk át (a DEBUG csak hexadecimális számokat fogad el). Az 5-ös megszakítás decimális címe 20, hexadecimálisan 14. Ezt a következő parancsokkal nézhetjük meg:

```
DEBUG
D 0000:0014 L 4
```

Az eredmény így néz ki:

```
54 FF 00 F0
```

Ha ezt szegmentált címmé alakítjuk, és figyelembe vesszük, hogy a szó bájtjai fordított sorrendben tárolódnak, akkor azt kapjuk, hogy a képernyő tartalmát kinyomtató megszakítási vektor tartalma F000 : FF54. Ugyanennek a DEBUG-parancsnak a segítségével bármelyik megszakítási vektor megvizsgálható.

3.1. táblázat

Az IBM személyi-számítógép családban használt fontosabb megszakítások

Megszakítás			Felhasználás
Decimális	Hexa-decimális	Cím	
0	0	0000	A CPU generálja, ha kísérlet történik 0-val való osztásra
1	1	0004	Lépésenkénti programvégrehajtás (pl. DEBUG)
2	2	0008	Nemmaszkolható megszakítás: a PCjr-en az NMI-nek különleges szerepe van
3	3	000C	Töréspontok programban történő elhelyezésére (pl. DEBUG)
4	4	0010	Aritmetikai túlcsorduláskor keletkezik
5	5	0014	Behívja a képernyő tartalmát kinyomtató szubrutint a BIOS-ból
8	8	0020	A hardver óraütése generálja
9	9	0024	A legtöbb modellnél a billentyűzeten lezajló esemény váltja ki; csak a PCjr szimulálja kompatíbilitási okokból
13	D	0034	A CRT függőleges képvisszafutása alatt generálódik a megjelenítés vezérlése céljából

Megszakítás			Felhasználás
Decimális	Hexa-decimális	Cím	
14	E	0038	A lemez hívja fel magára a figyelmet vele (pl. művelet befejezése)
15	F	003C	A nyomtató vezérlésénél használjuk
16	10	0040	A videoszolgáltatásokat hívja be a BIOS-ból
17	11	0044	A berendezések listáját megadó szolgáltatást hívja be a BIOS-ból
18	12	0048	A tár méretét megadó szolgáltatást hívja meg a BIOS-ból
19	13	004C	Lemezes szolgáltatásokat hív meg a BIOS-ból
20	14	0050	A kommunikációs szolgáltatásokat hívja meg a BIOS-ból
21	15	0054	A kazettás magnós szolgáltatásokat hívja meg a BIOS-ból
22	16	0058	A szabványos billentyűzetszolgáltatásokat hívja meg a BIOS-ból
23	17	005C	A nyomtatószolgáltatásokat hívja meg a BIOS-ból
24	18	0060	Elindítja a ROM-BASIC-et, vagy azt, ami helyette van
25	19	0064	A BIOS-ban meghívja az öntöltő indítóprogramot
26	1A	0068	A BIOS-ban meghívja az idő- és dátumszolgáltatást
27	1B	006C	A BIOS alatt ez a megszakítás keletkezik a *BREAK*-billentyű lenyomására; ha megírjuk ez a szubrutin hívódik meg
28	1C	0070	Ez a megszakítás minden óraütésnél keletkezik; ha megírjuk, meghívódik egy szubrutin
29	1D	0074	A video vezérlő paraméterek táblázatára mutat
30	1E	0078	A lemez alaptáblázatára mutat
31	1F	007C	A grafikus karakterek táblázatának 2. részére mutat
32	20	0080	DOS-ban behívja a programot befejező szolgáltatást
33	21	0084	Az összes DOS-funkció ezen keresztül hívódik meg
34	22	0088	Ha megírjuk, ez a szubrutin hívódik meg egy DOS alatt futó program befejezésekor
35	23	008C	Ha megírjuk, ez a szubrutin hívódik meg a DOS alatt megnyomott *BREAK*-billentyű hatására
36	24	0090	Ha megírjuk, ez a szubrutin hívódik meg a DOS alatt bekövetkezett kritikus lemezhiba hatására
37	25	0094	DOS-ból meghívja az abszolút lemezolvasó szolgáltatást
38	26	0098	DOS-ból meghívja az abszolút lemezre írás szolgáltatást
39	27	009C	A program befejeződik, de a DOS felügyelete alatt a tárban marad
68	44	0110	A grafikus karakterek táblázatának elejére mutat a PCjr-en
72	48	0120	Behívja azt a programot, amelyik a PCjr billentyűzetéből PC-billentyűzetet csinál
73	49	0124	A billentyűzetet kiegészítő berendezések transzlációs táblázatára mutat

A 3.1. táblázatban láthatók a legfontosabb megszakítási vektorok és címeik. A programozó számára valószínűleg ezek a megszakítások a leghasznosabbak. (☛ A legtöbb megszakításról további részletek találhatók a 8 – 13. fejezetig. A listán nem szereplő megszakítások többségét az IBM fejlesztési célokra tartalékolta.)

Megszakítási vektorok megváltoztatása

Programozási szempontból nem az a legérdekesebb a megszakítási vektorokkal kapcsolatban, hogy tartalmuk kiolvasható, hanem az, hogy megváltoztatva őket egy új megszakításkezelő szubrutinra mutatnak. Ezt úgy érjük el, hogy a megszakításkezelő szubrutint (ami eltér a ROM-BIOS vagy a DOS szabványos szubrutinjaitól) a RAM-ban tároljuk, majd a táblázat egyik létező megszakításához új címként ennek kezdőcímét rendeljük.

A vektort megváltoztathatjuk bájtonként, az assembly nyelv szintjén, vagy egy programozási nyelv megfelelő utasításával (pl. a BASIC POKE-jával). Néha fennáll annak a veszélye, hogy a megszakítás az átírási folyamat kellős közepén lép fel. Ha ezzel nem kell törődni, akkor használható a POKE-t tartalmazó eljárás. Különben két módszert használhatunk, hogy óvintézkedésekkel elejét vegyük a megszakítás bekövetkezésének, mialatt a vektorát megváltoztatjuk.

Az első módszernél kézzel változtatjuk meg a vektort, és a megszakításokat letiltó utasítással (CLI) függesztjük fel a megszakításokat a változtatás idejére. A CLI a nemmaszkolható megszakítás (NMI) kivételével az összes megszakítást letiltja. Eredetileg az NMI csak valóban sürgős események jelzésére szolgált (ég a gép), de sajnos felhasználták egészen közönséges eseményekhez is, pl. ha a PCjr-nél a billentyűkkel történik valami. Ennek az a következménye, hogy a CLI-utasítás alkalmazása ugyan meglehetősen biztos védelmet nyújt az ellen, hogy a változtatás közben lépjen fel megszakítás, de nem tekinthetjük tökéletesnek.

Két példán mutatjuk be az első módszert, amelyben a megszakítások letiltása után írjuk át a megszakítási vektort. Az első példában két MOV-utasítással töltjük fel a vektort:

```
XOR   AX,AX                  ;nullával töltjük fel a szegmensregisztert
MOV   ES,AX                  ;nullával töltjük fel a szegmensregisztert
CLI                          ;a megszakítások letiltása
MOV   WORD PTR ES:36,XX      ;a vektor relatív részének betöltése
MOV   WORD PTR ES:38,XX      ;a vektor szegmens részének betöltése
STI                          ;a megszakítások engedélyezése
```

Jóllehet, ez a módszer teljesen kézenfekvő, magában rejti annak a veszélyét, hogy a két MOV-utasítás között egy NMI következik be, igaz, ennek az esélye nagyon kicsi. A kockázat csökkenthető, ha a két MOV-ot egyetlen ciklikus vagy karakterlánc MOV-val helyettesítjük (MOVS). A ciklikus MOV használata jóval körülményesebb, mivel egy sor regisztert kell hozzá beállítani. Mégis felhasználjuk, főleg azért, hogy egy másik kódolási módot is bemutassunk, ami az elsővel azonos eredményt ad.

```
;először is töltsük fel a ciklikus mozgatáshoz szükséges regisztereket
XOR   DI,DI                  ;nullát ad eredményül
MOV   ES,DI                  ;a cél bekezdése = 0
MOV   DI,36                  ;a cél relatív címe
MOV   SI,XXXX                ;a forrás bekezdése
MOV   CX,2                   ;a szavak száma
CLD                          ;az „előre” irány beállítása
;az átmozgatás alatt a megszakítást letiltjuk
CLI                          ;a megszakítás letiltása
REP   MOVSW                  ;a szavak ciklikus átvitele
STI                          ;a megszakítások újraengedélyezése
```

Két módszert is bemutattunk arra, hogy lehet „csináld magad” módszerrel megváltoztatni a megszakítási vektort. Egy másik úton ezt a DOS 37-es szolgáltatásával is elérhetjük – ezt éppen erre találták ki. Két igen lényeges előnye van annak, ha a DOS-ra hagyjuk a megszakítá-

sok megváltoztatását. Az egyik az, hogy a DOS a lehető legbiztonságosabb módon fogja felülírni a megszakítási vektort. A másik már hosszabb távú. Abban a pillanatban, hogy az AT-modellnél megjelent a 80286-os IC, a PC-családban alapvetően megváltozott az olyan ismerős fogalmak jelentése, mint pl. megszakítási vektor vagy szegmensregiszter.

Amikor DOS-szolgáltatással változtatunk meg egy megszakítási címet, nem teszünk mást, mint csökkentjük annak esélyét, hogy programunk nem lesz kompatíbilis egy új géppel vagy egy új operációs rendszerrel.

Egy megszakítási vektor beállítása a DOS 37-es szolgáltatásával:

```
MOV     DX,XX               ;a vektor relatív része
MOV     DS,YY               ;a vektor szegmens része
MOV     AH,37               ;a megszakítás beállításának kódszáma
MOV     AL,9                ;a 9-es megszakítást változtatjuk meg
INT     33                  ;a DOS hívása
```

A példa a lehető legtisztábban mutatja a DOS-szolgáltatások használatát. Egy lényeges nehézség azonban rejtve maradt: a DOS-nak átadandó címet a *DS*-regiszterbe (adatszegmens) kellett tölteni; ez lehetetlenné teszi, hogy a megszokott módon (a *DS*-regiszteren keresztül) férjünk hozzá adatainkhoz. A probléma megkerülése nem kis erőfeszítést igényel. Egy lehetséges megoldás (a példa a Norton Utilities programban található):

```
PUSH    DS                        ;mentsük az aktuális adatszegmenst
MOV     DX,OFFSET PGROUP:XXX      ;a vektor relatív része
PUSH    CS                        ;kódszegmensünket mozgassuk át
POP     DS                        ;az adatszegmensbe
MOV     AH,37                     ;a DOS-szolgáltatás kódszáma
MOV     AL,9                      ;a 9-es megszakítást változtatjuk meg
INT     33                        ;a DOS hívása
POP     DS                        ;az eredeti adatszegmens visszaállítása
```

A legfontosabb címek a központi tár alján

A PC műkűködését nagy részben azok az adatok vezérlik, amelyek a központi tár alacsony című részén (különösen a hexadecimális 400-on, ill. 500-on) kezdődő két 256 bájtos területen találhatók. A két terület a BIOS-ból tölti fel a bekapcsolási eljárás. Annak ellenére, hogy ez a BIOS számára fenntartott terület, mi is megvizsgálhatjuk, esetleg meg is változtathatjuk az itt található adatokat. Akkor is tanulmányozhatjuk a BIOS vezérlőterületét, ha nem szándékozunk felhasználni, mert sok mindent tisztábban látunk majd (pl. a PC-család működését).

A félreértések elkerülése végett felhívjuk a figyelmet rá, hogy a 400-as tárcímet szegmentált formában felírhatjuk 0040:0000-ként vagy 0000:0400-ként is. Mindhárom jelölés ugyanazt a tárrekeszt jelöli.

A vezérlőadatok területe

A hexadecimális 400-as és 500-as mező néhány címe különösen érdekes. Legtöbbjük az egyes BIOS vagy DOS szubrutinok szempontjából létfontosságú adatokat tartalmaz. Sok esetben programjaink egy BIOS megszakítás meghívásával juthatnak az itt tárolt adatokhoz; a közvet-

len hozzáférés mindig lehetséges. Saját számítógépén mindenki könnyen ellenőrizheti e tárolók tartalmát a DEBUG vagy a BASIC segítségével.

A DEBUG-ot a következő parancsokkal használhatjuk:

```
DEBUG
D 0:XXXX L 1
```

A fenti példában *XXXX* annak a rekesznek a hexadecimális címe, aminek a tartalmát meg akarjuk vizsgálni. Az *L 1* közli a DEBUG-gal, hogy 1 bájtot akarunk látni. Kettőt vagy többet úgy nézhetünk meg, hogy az *L* után a megfelelő számot írjuk be.

A következő program segítségével tudunk BASIC nyelven adatokat megvizsgálni. Értelemszerűen a hexa *address.in.hex* és a decimális *number.of.bytes* helyére a megfelelő adatokat kell beírni:

```
10 DEF SEG = 0
20 FOR I = 0 TO number.of.bytes – 1
30    VALUE = PEEK (&Haddress.in.hex + I)
40    IF VALUE < 16 THEN PRINT "0"; 'vezető nullák miatt kell
50    PRINT HEX$ (VALUE); " ";
60 NEXT I
```

A leghasznosabb címek a következők (minden cím hexadecimális):

410 (2 bájtos szó). Ebben a szóban található a berendezések listáját tartalmazó adat. Ezt közli a 17-es (hexa 11, berendezések listája) szolgáltatás. A szó formátuma a 3.2. táblázatban

3.2. táblázat

A hexa 410-es címen található berendezéslista kódolása

Bit																Jelentés
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
X	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	Az installált nyomtatók száma
.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	Ha soros nyomtató is van installálva, akkor 1
.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	Ha játék adapter van installálva, akkor 1
.	.	.	.	X	X	X	.	.	.	.	.	.	.	.	.	Az RS – 232 soros portok száma
.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	Ha a DMA IC installálva van akkor 0; a DMA szabványos tartozék a PCjr-en kívül minden modellen
.	.	.	.	.	.	.	.	X	X	.	.	.	.	.	.	+ 1 = hajlékonylemezes meghajtók száma: 00 = 1 meghajtó, 02 = 2 meghajtó, 10 = 3 meghajtó, 11 = 4 meghajtó (l. 0-s bit)
.	.	.	.	.	.	.	.	.	.	X	X	.	.	.	.	Kezdeti megjelenítési mód: 01 = PCjr 40 oszlopos, színes 10 = 80 oszlopos színes, 11 = 80 oszlopos monokrom a többi modellnél
.	.	.	.	.	.	.	.	.	.	.	.	X	X	.	.	A regiszterkártyán található RAM mennyisége: 00 = 16K, 01 = 32K, 11 = 64K (ez a szokásos az összes modellnél, a 256K-s rendszerkártyák valószínű installálásának ellenére
.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	Nem használt, értéke 0
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	Ha van lemezmeghajtó, akkor 1 (ebben az esetben l. a 6-os és 7-es bitet)

látható. Ebben a formában a PC-ken és az XT-ken találjuk, egyes részei az újabb modelleken (a PCjr-en) eltérők lehetnek.

JR **412** (1 bájt). Ezt a bájtot csak a PCjr használja. Azt számlálja, hogy hány hibát észlelt a billentyűzet infravörös csatolója. Más modelleknél ezt a bájtot csak az inicializálás alatt használják. Érdekes bájt, de programozási szempontból nincs jelentősége.

413 (2 bájtos szó). A felhasználható memória méretét tartalmazza K egységekben. A PCjr-en az a méret, ami a képernyő-memória lefoglalása után fennmarad. A többi modellnél a szó jelentése kissé más: a teljes méretet jelenti. Modelltől függetlenül a szó azonos célt szolgál: megmondja, mennyi a felhasználható memória. A BIOS 18-as (hexa 12) szolgáltatása ennek a szónak az értékét adja meg.

417 (a billentyűzetállapot bitjeinek két bájtja). A ROM-BIOS szubrutinok e bájtok intenzív használatával vezérlik a billentyűzeten végbemenő események értelmezését. Megváltoztatásuk tulajdonképpen a lenyomott billentyűk értelmét változtatja meg. Az elsőt tetszés szerint meg lehet változtatni, a második megváltoztatása nem tanácsos. JR A PCjr-nél egy harmadik bájt is van a 488-as címen. (☛ A 6.3., 6.4. és a 11.2. táblázatban található az egyes bitek jelentése.)

419 (1 bájt). Ez a bájt alternatív billentyűzet számára van fenntartva. Felhasználását a jövőben tervezik.

41A (2 bájtos szó). A szó a 41E címen a BIOS billentyűzetpufferének elejére mutat. Felhasználásig ott vannak a lenyomott billentyűk kódjai.

41C (2 bájtos szó). A szó a puffer végére mutat.

41E (32 bájt, 16 db 2 bájtos egységként használják). A billentyűzetpuffer 16 billentyű kódját tárolja, míg ki nem olvassák a 22-es (hexa 16) BIOS-szolgáltatással. Ez egy ciklikus sorpuffer, ezért kell két mutató a kezdet és a vég jelölésére. (41A és 41C). Nem tanácsos ezekhez az adatokhoz nyúlni.

43E (1 bájt). Azt mutatja, hogy szükséges-e a hajlékonylemez újrakalibrálása a sávkeresés előtt. A 0., 1., 2., 3. bit a 0., 1., 2., 3. meghajtónak felel meg. Ha egy bit értéke 0, akkor szükséges az újrakalibrálás. Általában akkor találunk 0-t egy bit helyén, ha valami probléma volt a meghajtó legutolsó használatakor. Az újrakalibrálást jelző bit pl. 0 lesz akkor, ha **DIR**-parancsot adunk ki egy olyan meghajtóra, amelyben nincs lemez, majd az alábbi üzenetre *A*-t válaszolunk:

Not ready reading error B:
Abort, Retry, Ignore?

43F (1 bájt). A bájt a meghajtó motorjának állapotát mutatja. A 0., 1., 2., 3. bit a 0-s, 1-es, 2-es, 3-as meghajtónak felel meg. Ha egy bit értéke 1, a megfelelő motor jár.

440 (1 bájt). Ebben a bájtban a számláló a meghajtó motorjának kikapcsolásáig hátralevő időt mutatja. A számláló 37-ről indul (kb. 2 s) minden lemezes műveletnél, és minden óraütésnél eggyel csökkent. Ha a számláló eléri a nullát, a motor leáll.

441 (1 bájt). A bájt a lemezállapot információját tartalmazza. Minden bit más típusú hibát jelez. (L. a 3.3. táblázatot.) Minden egyes bit értéke 1, ha a hiba előfordult; és 0, ha nem volt hiba.

442 (7 bájt). Ez a 7 bájt a lemezvezérlőket leíró információkat tartalmaz.

Hexadecimális 449-nél a megjelenítés 30 bájtos vezérlő területe kezdődik. A BIOS itt tárolja a legfontosabb információkat a megjelenítésről. A programok nyugodtan megvizsgálhatják az itt tárolt adatokat, megváltoztatásuk azonban legtöbbször kockázatos. Bármelyik adat megváltoztatása hibát okozhat a gép működésében – a kísérletek néha látványosan bizarr

3.3. táblázat

A hexa 441-es címen található hajlékonylemez-státusz kódolása

A bit helye Jelentése 7 6 5 4 3 2 1 0	
X	Időtúllépés a hajlékonylemeznél; nem érkezett időben válasz
. X	Sikertelen sávkeresési művelet
. . X	A hajlékonylemez vezérlő IC-jének hibája
. . . X	A CRC-t ellenőrző összeg hibás: adathiba
. . . . X . . .	DMA-lemezhiba
. X . .	Nem találta a szektort: a lemez hibás, vagy nincs megformázva
. X .	A hajlékonylemezen nem találta a címjelet
. X	Hibás lemezparancs érkezett

3.4. táblázat

A hexa 449-es címen található megjelenítési mód kódolása

Kód	Jelentése
0	40 oszlopos szöveg, szín nincs
1	40 oszlopos szöveg, 16 szín
2	80 oszlopos szöveg, szín nincs
3	80 oszlopos szöveg, 16 szín
4	Közepes felbontású grafika, 4 szín
5	Közepes felbontású grafika, szín nincs (4 szürke tónus)
6	Nagy febontású grafika, 2 szín
7	Monokrom adapter megjelenítési módja
8	Durva felbontású grafika, 16 szín (a szabványos színes-grafikus adapteren nincs)
9	Közepes felbontású grafika, 16 szín (a szabványos színes-grafikus adapteren nincs)
10	Nagy felbontású grafika, 4 szín (a szabványos színes-grafikus adapteren nincs
13	Közepes felbontású grafika, 16 szín (a szabványos színes-grafikus adapteren nincs)
14	Nagy felbontású grafika, 16 szín (a szabványos színes-grafikus adapteren nincs)
15	Különlegesen nagy felbontású grafika, 4 szín (a szabványos színes-grafikus adapteren nincs)

eredményeket produkálnak. Mindössze a kurzor helyét jelölő mezőkről mondhatjuk el, hogy megváltoztatásuk veszélytelen és hasznos is. (☛ A kurzorról részletesebben lesz szó a 450H címnél.)

449 (1 bájt). A bájtban található 0 és 10 vagy 13 vagy 15 közötti érték szabja meg a megjelenítési módot. (☛ L. a 3.4. táblázatot.) A megjelenítési módok kódolása megegyezik a BIOS-nál használttal. (☛ A 9. fejezetben foglalkozunk bővebben a videoszolgáltatásokkal, a 4.1. táblázatban általános információ található a megjelenítési módokról.)

EGY BASIC-program a következő utasításokkal olvashatja ki ezt a bájtot:

```
DEF SEG = 0                          'DS regiszter = 0
VIDEO MODE=PEEK (&H449)              'nézd meg a 449-es hexa címet
```

(☛ A 4. fejezetben külön is tárgyaljuk a BASIC 4-es és 5-ös módját.)

44A (2 bájtos szó). Ebben a szóban található, hogy egy képernyősorba hány karakter fér. 20, 40 vagy 80 állhat itt hexadecimális formában (20 oszlopos a 8-as, alacsony felbontású grafikájú video mód).

44C (2 bájtos szó). A képernyő felfrissítésének hossza. Az egy képernyőn található karakterek száma a videomódtól függően változó.

44E (a bájtos szó). A képernyő kezdetének relatív címe. A jelenleg látható kép relatív kezdőcíme a videomemórián belül. Valójában a relatív cím megadásával azt mutatja ez a szó, hogy melyik látható képet használjuk.

450 (8 db 2 bájtos szó). Ezek a szavak a kurzor helyét adják meg a 8 látható lapon – a 0-ás lappal kezdve. Az első bájt mindig az oszlop (0-tól 19-ig, 39-ig vagy 79-ig), a második bájt pedig a sor (0-tól 24-ig). Az itt található adat megváltoztatásával megváltozik a kurzor pozíciója. Olyan programozási nyelveken, amelyekben nincsenek kurzorvezérlő utasítások, ezzel kényelmesen beállíthatjuk a kurzort anélkül, hogy assembly nyelvű illesztő szubrutinokat kellene írni a BIOS-hoz.

Ha egy bájtot megváltoztatunk, akkor a változtatás csak a *következő* képernyőre íráskor válik észrevehetővé. Ennek szemléltetésére indítsuk el a DEBUG-ot, és írjuk be ezt a parancsot:

```
F 0:450 L 2 8 8
```

A *RETURN* lenyomása után a kurzor a 8. sor 8. oszlopára ugrik. Nyilvánvalóan ez nem nevezhető jó programozási technikának, de egyszer még jól jöhet az ismerete.

460 (2 bájtos szó). Ez a két bájt a kurzor méretét tartalmazza, amit a kurzor által befutott sorok számából határoz meg. Az első bájtban van a kurzor utolsó sorának száma, a második-ban a kurzor első sorának száma. A kurzor helyét meghatározó bájtoktól eltérően ezeket hiába változtatjuk meg, a kurzor ettől automatikusan nem változik meg.

462 (1 bájt). Ebben a bájtban van az aktuális képernyőoldal sorszáma.

463 (2 bájtos szó). Ebben a szóban található a 6845-ös videovezérlő IC portcíme. Értéke rendszerint hexadecimális 3D4.

465 (1 bájt). Ebben a bájtban van a CRT (képernyő) megjelenítési módja.

466 (1 bájt). A bájt a színpaletta maszkbitjeit tartamazza. (☛ A palettáról a 4. fejezetben még fogunk olvasni.)

467 (5 bájt). Ez az 5 bájt a kazettás magnetofon vezérlésére való.

46C (4 bájt, 2 db 2 bájtos szóként tárolva, de egy 4 bájtos számként kezelve). Ez a terület az időmérés fő területe. Tartalma minden óraütésnél eggyel növekszik. A gép úgy kezeli, mintha éjfélkor nulláról indulna. Amint a számláló eléri a 24 órának megfelelő értéket, a 0 és a 470-es című bájt 1 lesz. A DOS vagy a BASIC ebből az értékből kiindulva számítja ki az időt, az idő beállításakor pedig a megfelelő számlálóérték ide iródik. Ezt az értéket a BIOS 26-os (hexa 1A) megszakítása olvassa ki vagy állítja be.

470 (1 bájt). Ez a bájt azt jelzi, hogy az óra áthaladt éjfélen. Amint az óra elhagyja az éjfélt – és visszaáll nullára –, ez a bájt 1 lesz, ami azt jelenti, hogy a dátumot meg kell növelni. Az

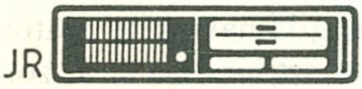
értéket az óraütést figyelő szubrutin állítja be, hogy jelezze: éjfél elmúlt. Akkor áll vissza nullára, amikor az órát a 26-os (hexa 1A) megszakítással kiolvassák. Az automatikus nullázás azon a feltételezésen alapul, hogy az időt kiolvasó összes program megnöveli a dátumot, ha észleli ezt a jelzést.

MEGJEGYZÉS: Éjfélkor a bájt értéke 1 lesz, nem pedig eggyel növekszik. Semmi sem jelzi, ha az óra kiolvasása nélkül kétszer is elmúlna éjfél.

471 (1 bájt). A BIOS-on belül ez jelzi a *BREAK*-billentyű lenyomását. Ha a 7. bit értéke 1, akkor a *BREAK*-billentyűt lenyomták. (A *BREAK* esetleg több billentyű kombinációja.)

472 (2 bájtos szó). Ezek a bájtok azt jelzik, hogy a billentyűzetről kezdeményezett rendszer töltése folyik. Ha a billentyűzetről a rendszer töltését indítjuk a *Ctrl-Alt-Del* billentyűkombinációval, a szó értéke hexadecimális 1234 lesz, míg a betöltés folyik. Valódi ritkaság ez a szó!

473 (1 bájt). Az IBM számára fönntartott bájt. Értéke hexa 24 lesz, ha a részvény osztalékot hoz.



474 (4 bájt). Ezt a 4 bájtot kizárólag a PCjr használja lemezvezérlésre.

478 (8 bájt, 2 db 4 bájtos mező). Ezeket a bájtokat csak a PCjr használja, abból a célból, hogy a párhuzamos nyomtatónál és a soros portnál (vagy a soros nyomtatónál) az időtúllépést figyelje.

485 (1 bájt). Ebben a bájtban található az a karakter, amit az ismétlőbillentyű lenyomásakor ismételni kell. A PCjr sajátossága.

486 (1 bájt). Ezt a bájtot a billentyű ismétlése előtti kivárási idő mérésénél használják. A PCjr sajátossága.

487 (1 bájt). Ebben a bájtban tárolódik az aktuális *Fn* funkciókód. A PCjr sajátossága.

488 (1 bájt). Ezt a bájtot a PCjr a billentyűzet 3. állapotbájtjaként használja. A 417-en és 418-on található másik két állapotbájtot minden modell használja, köztük a PCjr is. (☛ A bitek jelentését l. a 6. fejezetben, a 6.2. és a 6.3. táblázatban.)

500 (1 bájt). A DOS és a BASIC a képernyő kinyomtatásának vezérlésére használja ezt a bájtot. A rekesznek három értéke lehet. Ezek hexadecimálisan a következők:

00, ha minden rendben;
01, ha a képernyő kinyomtatása folyik;
FF, ha a nyomtatás közben hiba volt.

504 (1 bájt). Ezt a bájtot akkor használja a DOS, ha egy egylemezes rendszer – pl. az XT vagy a PCjr – kétlemezes rendszert imitál. Az itt tárolt érték azt jelzi, hogy a valódi meghajtó *A* vagy *B* egységként működik-e. A következő értékeket használja:

00, ha *A* egységként működik;
01, ha *B* egységként működik.

510 (2 bájtos szó). Itt tárolja a BASIC az adatszegmens (*DS*) alapértékét. Ez a BASIC feltételezett adatszegmensének mutatója.

A BASIC lehetővé teszi, hogy a **DEF SEG**-érték utasítással tetszőleges értéket írjunk be az adatszegmensbe. (A szegmensen belüli relatív címet a **PEEK**- vagy a **POKE**-utasításban adjuk meg.) Az adatszegmenst visszaállíthatjuk az eredeti értékére, ha a **DEF SEG**-utasítást használjuk. Bár a BASIC nem szívesen árulja el az itt tárolt adatszegmens értéket, a következő utasításokkal mégis megtudhatjuk:

```
DEF SEG = 0
DATA.SEGMENT = PEEK(&H511)*256+PEEK(&H510)
```

MEGJEGYZÉS: A BASIC az adatszegmens-regiszter alapértékének megfelelően kezeli saját belső adatait. Ha ezt megpróbáljuk megváltoztatni, akkor valószínűleg lehetetlenné tesszük a BASIC helyes működését.

512 (4 bájt): A BASIC ezt a területet a saját óra szubrutinjára mutató megszakításvektorként használja.

🕮 MEGJEGYZÉS: A nagyobb teljesítmény érdekében a BASIC a szokásos ütem négyszeresével járatja a rendszerórát. Ezért kell a BASIC-nek a BIOS megszakítási szubrutinját a sajátjával helyettesítenie. A szabványos BIOS megszakítási szubrutint a BASIC normális ütemben hívja meg – azaz négy gyors órajelenként egyszer. (☛ E témáról további részletek a 7. fejezetben olvashatók.)

516 (4 bájt). A BASIC ezt használja a BREAK-et kezelő szubrutin megszakításvektoraként.

51A (4 bájt). Ezt használja a BASIC a hajlékonylemez hibáit kezelő szubrutin megszakításvektoraként.

A programon belüli kommunikációs terület

Bár a 400-zal kezdődő terület legnagyobb részét a BIOS-t vezérlő adatok töltik ki, található itt egy programon belüli kommunikációs terület is, az ún. ICA (Intra-application Communications Area). Az ICA számára fenntartott 16 bájt 4F0 és 4FF között található. Feladata olyan adatok tárolása, amin több program is osztozhat. Különösen hasznos olyan programoknál, amelyek külön DOS-programokként futnak, de üzenetet kell hagyniuk a programegyüttes más tagjai részére. Az ICA-t viszonylag ritkán használják. Az ICA-t használja pl. az IBM aszinkron kommunikációs programja (Asynchronous Communications), a Lifetree Volkswritere és a szerző TimeMarkja.

Mivel az ICA-ban tetszőlegesen sok program tárolhat adatokat, előfordulhat, hogy egyszerre több program adatát tartalmazza. Ez azzal járhat, hogy egyes adatok felülíródnak. Az ICA területének használatakor javasoljuk egy ellenőrző összeg és egy azonosító „aláírás" beírását, amivel azonosítani lehet, hogy az ICA-ban talált adat a miénk, és más program nem változtatta meg.

✷ FIGYELEM! *Az ICA egészen biztosan 4F0 és 4FF közötti 16 bájton van. Egyes IBM Technical Reference kézikönyvekben nyomdahiba miatt 500, ill. 5FF szerepel. Ez tévedés!*

A ROM-változat és a géptípus azonosítója

Mivel a BIOS be van égetve a tárba, nem könnyű bővíteni vagy javítani. A ROM-programokat tehát nagyon alaposan ki kell próbálni, mielőtt a tároló IC-kbe beírják. Annak ellenére, hogy súlyos hiba könnyen bekerülhet a rendszer ROM-programjaiba, az IBM igen jó statisztikával dicsekedhet. Eddig mindössze jelentéktelen kis hibákat találtak a PC-család ROM-programjaiban, és az új gépeken ezeket ki is javították.

A ROM-szoftver eltérő változatai némi fejtörést okozhatnak, ha a programozó azt tapasztalja, hogy az eltérés hatással van a programok futására. Ennél nagyobb nehézséget jelenthet, hogy a család egyes tagjainál – különösen a PCjr-nél és az AT-nél – a ROM-BIOS-szubrutinok eltérnek a szabványos PC-ben levőktől.

A megfelelő ROM-programok és a megfelelő gépek ellenőrizhetősége érdekében az IBM a tár két végére azonosítót helyezett el. Az egyik a ROM kibocsátásának a dátuma – ezzel a BIOS-változat azonosítható –, a másik a modell azonosítója. Az eredeti IBM gépekben ezek a jelzések mindig megtalálhatók, és a teljes IBM-család gyártói közül is átvették néhányan.

A ROM kibocsátásának dátuma egy 8 bájtos területen található az F000:FFF5 és az F000:FFFC között, ami két bájttal megelőzi a típusazonosító kódot. A dátum ASCII-karakterekkel, a megszokott amerikai formában van beírva; pl. 06/01/83 – ez 1983. június 1-jét jelent. A kibocsátási dátum minden IBM személyi számítógép közös jellemzője, de csak néhány kompatíbilis gépen található meg. A Compaqnál pl. nincs, a Panasonic Senior Partnernél viszont van.

A kibocsátási dátumot csak a ROM különböző változatainak azonosítására lehet felhasználni. (L. a 3.5. táblázatot.) Csak olyankor használjuk, ha a feladat megköveteli, hogy a program másképp működjön a különböző ROM-ok esetében. (Valószínűleg inkább a gép típusának azonosítójára van szükség, ha azt akarjuk, hogy egy program felkészüljön az egyes modellek sajátosságaira.)

A DEBUG segítségével a következő parancsokkal nézhetjük meg a kibocsátási dátumot:

```
DEBUG
D F000:FFF5 L 8
```

3.5. táblázat

A különböző ROM-változatok kibocsátási dátuma

Típusjelzés	Gép
04/24/81	Eredeti PC
10/19/81	Javított PC (néhány hibát kijavítottak)
08/16/82	Eredeti XT
10/27/82	A PC feljavítása az XT BIOS szintjére
11/08/82	Eredeti Portable PC
06/01/83	Eredeti PCjr
01/10/84	Eredeti AT

Vagy a következő programot használjuk:

```
10 DEF SEG=&HF000
20 FOR I=0 TO 7
30    PRINT CHR$(PEEK(&HFFF5+I));
40 NEXT
50 END
```

Pl. a következő adatok találhatók a szerző 3 PC-jében: az egyik dátuma 04/24/81, a másiké 10/19/81, a harmadiké pedig 10/27/82.

Bizonyos körülmények között meg lehet szerezni a BIOS továbbfejlesztett változatát: annak a PC-bővítőegységnek, amelyik a PC-t XT-vé alakítja 10/27/82 a dátuma.

A gép azonosítója az F000:FFFE című bájton található. A 3.6. táblázat 5 db IBM PC-modell azonosítóját mutatja be. Várható, hogy újabb modelleknél a minta folytatódni fog.

Figyeljünk, mert némi következetlenség tapasztalható az azonosítók kiosztásában. Eredetileg az FE értéket az XT, majd később a Portable PC azonosítójának szánták. Valójában mégis sok XT-ben a PC azonosítója az FF. Általában sem tekinthetjük az azonosítók kiosztását

3.6. táblázat
IBM-PC modellek gépazonosítója

Azonosító		Gép
Decimális	Hexadecimális	
255	FF	PC (az eredeti IBM személyi számítógép)
254	FE	XT és Portable PC
253	FD	PCjR
252	FC	AT

3.7. táblázat
Két Compaq-modell nem hivatalos gépazonosítója

Azonosító		Gép
Decimális	Hexadecimális	
45	2D	Compaq (PC-vel egyenértékű)
154	9A	Compaq-Plus (XT-vel egyenértékű)

teljesen megbízhatónak; egyes azonosítókról az IBM sok mindent elmondott, és sok mindent megvalósított; mégsem mindig az lett az azonosító, amit előzőleg annak nevezett. Van azonban egy egyszerű szabály az azonosítókkal kapcsolatban. Ha a gépek között olyan nagy a különbség, hogy egy programnak teljes biztonsággal meg kell állapítania, melyik gépről van szó, akkor az azonosító sziklaszilárd támaszt nyújt. Ennek alátámasztására példa a PCjr és az AT: ezeknek valóban eltérő jellegzetességei vannak. Ha a modellek közötti eltérés kisebb – mint pl. az eredeti PC, a szabványos PC, a PC-2 (ennek 256K memóriája lehet a rendszerkártyán), az XT és a Portable PC esetében –, akkor az azonosító bizonytalanabb. Gyakorlatilag úgy vehetjük, hogy az FF és az FE ugyanazt a gépet azonosítja: a többé-kevésbé szabványos PC-t.

Lehetséges, hogy IBM kompatíbilis számítógépek ugyanilyen módon azonosíthatók, azonban nem ismerünk erről megbízható, hivatalos közleményt. A szerző programjai az első két Compaq-gép azonosítóját felismerik, de ezt nem lehet hivatalosnak tekinteni.

A DEBUG segítségével a következő parancsokkal nézhetjük meg a gép azonosítóját:

```
DEBUG
D F000:FFFE L 1          'a megadott címtől kezdve 1 bájtot ír ki.
```

Programmal pl. a következő módszerrel vizsgálható meg ez a bájt:

```
10 DEF SEG=&HF000   'beállítja az F000 szegmenscímet
20   IF PEEK(&HFFFE)=253THEN PRINT   "Azt hiszem, Junior vagyok"
30   IF PEEK(&HFFFE)=254THEN PRINT   "Azt hiszem, XT vagyok"
40   IF PEEK(&HFFFE)=255THEN PRINT   "Azt hiszem, PC vagyok"
50   IF PEEK(&HFFFE)=252THEN PRINT   "Azt hiszem, AT vagyok"
60 END
```

A ROM-BASIC

Most rátérünk a ROM harmadik részére: a ROM-BASIC-re. A ROM-BASIC szerepe kettős. Először is ez az alapja a BASIC nyelvnek, tartalmazza a parancsok zömét, és a BASIC által használt legalapvetőbb dolgokat, pl. a memória kezelését. A BASIC lemezes változata – ezt tartalmazza a BASIC.COM és a BASICA.COM lemezes fájl – tulajdonképpen csak adalék a ROM-BASIC-hez. A munka javát a ROM-BASIC látja el. A ROM-BASIC másik feladata, hogy az IBM által „kazettás"-nak nevezett BASIC-et biztosítsa. Ez a BASIC jelentkezik be, ha lemez nélkül hívunk rendszert a gépen.

Bármelyik értelmezős BASIC-et is használjuk: a kazettás BASIC-et, a PCjr cserélhető BASIC-ét vagy valamelyik lemezes (BASIC vagy BASICA) BASIC-et; a ROM-BASIC is működik – bár ennek semmi jelét nem tapasztaljuk. Másrészt a lefordított BASIC programok nem használják a ROM-BASIC-et.

A ROM-BASIC az IBM saját PC-családjának jellegzetessége. A bővebb PC-család egyik tagjánál sem találkozunk a ROM-BASIC-kel. Ehelyett a ROM-BASIC-kel egyenértékű programrészek a megfelelő lemezes BASIC részét képezik.

A ROM bővítései

A ROM negyedik alkotórészének több köze van a PC felépítéséhez, mint a tár tényleges tartalmának. A PC tervezésekor a beépített ROM-programok kétféle bővítésével számoltak; a ROM-BIOS-szoftver permanens (állandó jellegű) bővítésével és a cserélhető ROM-tárakkal megvalósított bővítéssel. Mindkettő számára külön tárterületeket tartottak fenn.

A ROM-BIOS permanens bővítései olyan programokat tartalmaznak, amelyek a beépített ROM-BIOS-hoz teljesen hasonlóan működnek, azt további lehetőségekkel bővítik. Ezek általában új perifériák kiszolgáló programjai. Ilyen ROM-ra legjobb példa az XT-modellnél bevezetett merevlemezes ROM-BIOS-programja. Hasonló jellegű kiegészítés található az EGA-kártyán. Mivel az eredeti ROM-BIOS programozói nem írhatták meg a később megszületendő hardver kiszolgáló programjait, kézenfekvő, hogy a ROM bővítése szükséges és hasznos lehetőség.

A permanens ROM-BIOS-bővítések számára két tárterület van fenntartva. Egyik (a sajnos modellenként változó kihasználatlan rész) az F. tárblokkban. A legtöbb modellen rendelkezésre áll az F000 szegmensbekezdéstől az F600-ig terjedő 24K-s terület (a szerző egyik PC-jének nem IBM-től származó merevlemezes ROM-BIOS-a az F400-as paragrafusban található). A ROM-bővítések számára fenntartott másik terület a C. blokkban van, a C000-s szegmensbekezdéstől CFFF-ig. Az IBM XT merevlemezének ROM-BIOS-a ezen a területen a C800-as szegmensbekezdéstől, az IBM EGA-kártyáé pedig a C000-s bekezdéstől található meg. Bár az IBM által szállított permanens ROM-bővítések helye kiszámítható, más gyártók termékeinél viszont mindig fönnáll a konfliktus lehetősége.

Általában a permanens ROM-bővítések majdnem szabályosan megvalósított bővítések: azaz vagy egy bővítőkártyán találhatók, vagy a rendszerkártya valamelyik üres ROM-aljzatába dugaszolják be.

A cserélhető szoftvert viszont úgy tervezték, hogy könnyen be lehessen helyezni és el lehessen távolítani. Általában a lemezhez hasonló módon használják; azaz megadott célra átmenetileg betölthetünk egy programot. A cserélhető programok számára egy nagy 128K-s terület van fenntartva: a teljes D. és E. blokk.

A ROM mindkét kibővítése a bekapcsolási eljárás során válik a ROM részévé. A bővítéseket az eljárás úgy találja meg, hogy a C000-s blokktól kezdve minden 2K-s blokkba megnézni, ott van-e a ROM-bővítésre jellemző azonosító (hex 55AA). Amint az azonosítót megtalálja, az indító szubrutin ideiglenesen átadja a vezérlést a ROM-bővítésnek, hogy az a bővítést beillessze a gép működésébe. Ezen a ponton a bővítés mindent megtehet, akár az egész gép működtetését is átveheti. Néhány cserélhető program ezt meg is teszi. Azonban a bővítés általában egyszerűen a rábízott berendezést ellenőrzi – pl. a merevlemez ROM-BIOS-a bekapcsolja a lemezt, és a tár alacsony című részén keresztül szimbolikusan kezet ráz a BIOS többi részével. A BIOS minden része így tisztában lesz a partnereivel. Amint az inicializálást befejezte, a ROM-bővítés rendszerint visszaadja a vezérlést a BIOS fő részének, és az befejezi a gép beindítási proceduráját.

Cserélhető szoftver

Mivel a ROM-bővítésekről nem kapunk teljes képet a cserélhető szoftver (Software Cartridges) ismertetése nélkül, egy rövid bekezdés erejéig legalább a legalapvetőbb dolgokat ismertetjük. Ennél részletesebb információ található az Exploring the IBM PCjr-ben és az IBM Technical Reference kézikönyvében.

A cserélhető szoftver beégetett programot tartalmaz. Minden PC-modellnél alkalmazható, amelynek megfelelő a kiképzése (pl. PCjr). Egy ilyen doboz kapacitása 2K-tól 64K-ig terjedhet, a hardver tervezésétől függően.

A doboz tartalma a 6 tárhely bármelyikén megjelenhet, a ténylegesen választott cím a programba van beleírva. Szegmensbekezdéses jelöléssel a 6 lehetséges cím: D000, D800, E000, E800, F000 és F800. A D. és az E. blokkban található 4 címet hagyományosan erre a célra használják. Az F. blokkban található 2 cím ROM-BIOS területét írja felül. Ez lehetővé teszi, hogy a cserélhető szoftver ideiglenesen átvegye a beépített ROM-BIOS szerepét.

A cserélhető szoftvernek szabványos fejblokkja van. Ezt a ROM-BIOS a program azonosítására és tartalmának meghatározására használja.

A fejblokk

A *fejblokk* (Cartridge Header) a ROM-bővítés szokásos 2 bájtos azonosítójával (hexa 55AA) kezdődik, utána egy 1 bájtos hosszkód áll. A ROM-bővítéseknél a hosszat 512 bájtos, vagyis 0,5K-s egységekben adják meg. Pl. egy 8K-s játék hosszkódja 16, egy 32K-s BASIC-é 64.

A hosszkódot az inicializálásra szolgáló utasítást tartalmazó 3 bájt követi. A mező számára fenntartott területen tetszőleges – legföljebb 3 bájt hosszú – utasítás tárolható. Rendszerint egy távolra visszatérő 1 bájtos **RET**- vagy **FAR RET**-, esetleg egy 3 bájtos **JMP**-utasítás van itt. Az itt tárolt utasítás célja az, hogy elindítsa az inicializálást, ha erre szükség van. Ha BASIC programról van szó (amihez tehát szükség van a cserélhető BASIC-értelmezőre), akkor a 3 bájt tartalma egészen különleges; a szokásos távolra visszatérő **FAR RET**-utasítás után (hexa CB) a ROM-bővítések azonosítója áll megfordítva (hexa AA55). Nehogy egy más típusú programot BASIC-programmal tévesszünk össze! Legyen a mező utolsó két bájtja – csakis BASIC-programoknál – AA55.

Az inicializáló mező után (a 7. bájton) egy DOS-tartalomjegyzék kezdődik. Ez azokat a DOS-parancsokat azonosítja, amelyeket a cserélhető szoftver esetleg tartalmaz. Ha van ilyen program, akkor ez a DOS teljes értékű belső parancsává válik, pont olyanná, mint pl. a **DIR**,

a COPY vagy a TIME. Ha ilyen program nincs, a tartalomjegyzék akkor sem hiányozhat; ilyenkor ún. *üres* tartalomjegyzék szerepel. A DOS-tartalomjegyzék a parancsnevek felsorolásából áll, amit egy 0 bájt követ. Az üres tartalomjegyzék egyszerűen egy 0 értékű bájtból áll.

A parancsnevek 3 mezőn adhatók meg. Egy 1 bájtos mező tartalmazza a parancsnév hosszát; a második mező a parancsnevet tartalmazza ASCII nagybetűkkel; a harmadik mezőn egy 3 bájtos ugróutasítás áll, amely a parancsot végrehajtó programra adja a vezérlést.

* FIGYELMEZTETÉS: *Egyes IBM Technical Reference kézikönyvekben hibásan 2 bájtos relatív szóként tüntetik fel a parancsnév utolsó mezejét. Valójában egy 3 bájtos ugróutasításról van szó.*

3.8. táblázat
A PCjr BASIC nyelv cserélhető szoftverének fej része. Figyeljük meg, hogy a parancsértelmezőben tárolt belső DOS-parancsok – DIR, TIME stb. – táblázata milyen hasonló az itt található DOS-parancs táblázathoz

Kezdőcím	Tartalom	Jelentése
0	55 AA	Aláírás
2	40	Hossz: hexa 40 = 64 > 1/2K = 32K
3	E9 1D 00	JMP 0023: ugrás az inicializáló kódra
6	05	A következő parancsnév hossza (5)
7	BASIC"	A parancs neve, ASCII-karakterekkel
12	E9 82 01	JMP 0191: ugrás a BASIC elejére
15	06	A következő parancsnév hossza (6)
16	„BASICA"	A parancs neve ASCII-karakterekkel
22	E9 78 01	JMP 0191: ugrás BASICA elejére (azonos a BASIC kezdetével)
25	00	Tartalomjegyzék vége

A fejblokkot a tényleges adatok követik, rendszerint gépi nyelvű programok. Ha BASIC-ben írt programról van szó, akkor ez éppen úgy – tömörített vagy „tokenizált" formában – van tárolva, mintha lemezen lenne. Ne feledkezzünk meg arról, hogy egy tömörített formában tárolt BASIC-program 1. bájtja a közönséges (hexa FF) vagy a védett (hexa FE) programot jelzi.

Megjegyzések

A ROM listájával köteteket lehetne megtölteni, és az IBM Technical Reference kézikönyvekben tényleg sok helyet foglal el. Az IBM nem veszi jó néven, ha valaki direkt módon felhasználja a ROM-listában talált adatokat, ennek ellenére esetenként hasznos és gondolatébresztő foglalatosság egy kicsit keresgélni benne. Mivel minden tőlünk telhetőt megtettünk a ROM-BIOS-szubrutinok kiemelésére, mielőtt valaki maga lát hozzá a kutatáshoz, érdemes ezt a részt átolvasnia.

4 Képernyős alapismeretek

Bevezetés • 62

A képernyőadapterek • 62

A tár- és a képernyőadapterek • 63
A kép előállítása • 63

Megjelenítési módok • 65

A képernyő felbontása • 66
A megjelenítési módok vezérlése • 67

A színek használata • 67

Színelnyomásos módok • 69
Színek a szöveges és a grafikus módoknál • 70

A képernyő-memória belseje • 74

Képernyőoldalak szöveges módoknál • 75
Képernyőoldalak grafikus módoknál • 76
Karakterek megjelenítése szöveges és grafikus módokban • 76

A képernyő vezérlése • 80

A hardver közvetlen vezérlése • 81

Kompatibilitási szempontok • 84

Bevezetés

Sok ember számára a képernyő jelenti magát a számítógépet. Gyakran egy programot pusztán a megjelenítés minősége és a vizuális tervezés színvonala alapján minősítenek. Ebben a fejezetben szemügyre vesszük, milyen képernyőket használ az IBM PC-család, és hogyan jön létre a kép. Még fontosabb megtanulni, milyen módon érhetjük el a kívánt képi effektusokat.

A képernyőadapterek

A kép előállításához a PC-család legtöbb tagjának – beleértve a PC-t, az XT-t és az AT-t – szüksége van egy ún. képernyőadapterre. Ez egy olyan áramköri kártya, amit a számítógép egyik bővítő csatlakozójába kell helyezni. A PCjr-hez, a Portable PC-hez és a Compaqhoz eleve tartozik képernyőadapter, bár az utóbbiak esetében az adapter cserélhető.

A képernyőadapter a CRT-vezérlő nevű IC-vel a számítógépet és a képernyős egységet kapcsolja össze. Az adapter ezenkívül tartalmaz több programozható I/O portot, egy ROM-ba írt karaktergenerátort és a megjelenített információ tárolására szolgáló RAM-ot.

Sokféle képernyőadapter létezik, de mindegyik mintája az IBM által a PC-hez tervezett két adapter egyike: a színes-grafikus vagy a monokrom adapter. Leginkább erről a két adapterről fogunk beszélni, a többire kiegészítő megjegyzésekkel utalunk.

A kép létrehozásának két, alapvetően eltérő módját különbözteti meg az IBM: a szöveges és a grafikus módot. A szöveges módban csak karakterek jeleníthetők meg, bár a karakterek segítségével egyszerű vonalas ábrák is létrehozhatók. (☛ A *C* függelék foglalkozik bővebben a karakterekkel.) A grafikus mód segítségével elsősorban bonyolult rajzok állíthatók elő, de alkalmas különböző formájú és méretű karakterek megjelenítésére is.

A színes-grafikus adapter szöveges és grafikus módban egyaránt működhet, de alkalmas különféle formátumú és színű ábrák, ill. karakterek megjelenítésére. Úgy tervezték, hogy minden megjelenítővel együtt tudjon működni, a szabványos tv-től a nagyfelbontású színes monitorig.

Ezzel ellentétben a monokrom adapter csak szöveges módban működhet, és a tárolt alfanumerikus és grafikus karaktereket csupán egyetlen színben képes megjeleníteni. A monokrom adaptert professzionális üzleti alkalmazásokhoz fejlesztették ki, így csak az IBM monokrom adapterrel (vagy ezzel egyenértékű berendezéssel) tud együttműködni. Ez egy. nagyfelbontású monitortípus. (☛ A következőkben részletesen foglalkozunk a monitorokkal.) Az üzleti élet, de sok professzionális felhasználó is előnyben részesíti a monokrom képernyőt a színes-grafikussal szemben, mert jobban olvasható. A monokrom képernyő választásával azonban lemondanak a színekről és a grafikáról, erről a két értékes képi eszközről.

A korlátok kiküszöbölése érdekében egyes hardvergyártók létrehozták az IBM monokrom adapterének módosított változatait. Ilyen pl. a népszerű Hercules képernyőadapter. Ez sikeresen ötvözi a színes-grafikus adapter grafikai lehetőségeit – színek nélkül – a monokrom adapter jó minőségű szövegmegjelenítésével. Ezen kívül tartalmaz néhány új szolgáltatást is. Az adapter grafikája jobb, mint a színes-grafikus adapteré. Az IBM EGA-adaptere hasonló módon állít elő grafikát a monokrom képernyőn.

Az összes PC-nek durván a kétharmada a szabványos monokrom adapterrel van felszerelve, ezért nincs rajta grafika és szín. Bár a színek és a grafika lényeges előnyöket nyújt, a legtöbb PC ezek nélkül is jól boldogul. Számítógépes programok tervezésekor tartsuk szem előtt, hogy a legtöbb számítógép és a legtöbb PC csak szöveget tud megjeleníteni.

A PC képalkotását úgy érthetjük meg a legjobban, hogy megismerkedünk az eredeti színes-grafikus adapter jellemzőivel, és külön megjegyezzük, ahol a monokrom adapter ettől eltér. (Az eltérések általában jelentéktelen részletek.) Bemutatjuk azt is, hogyan fejlesztették tovább a színes-grafikus adaptert az EGA-nál és a PCjr-nél.

A tár- és a képernyőadapterek

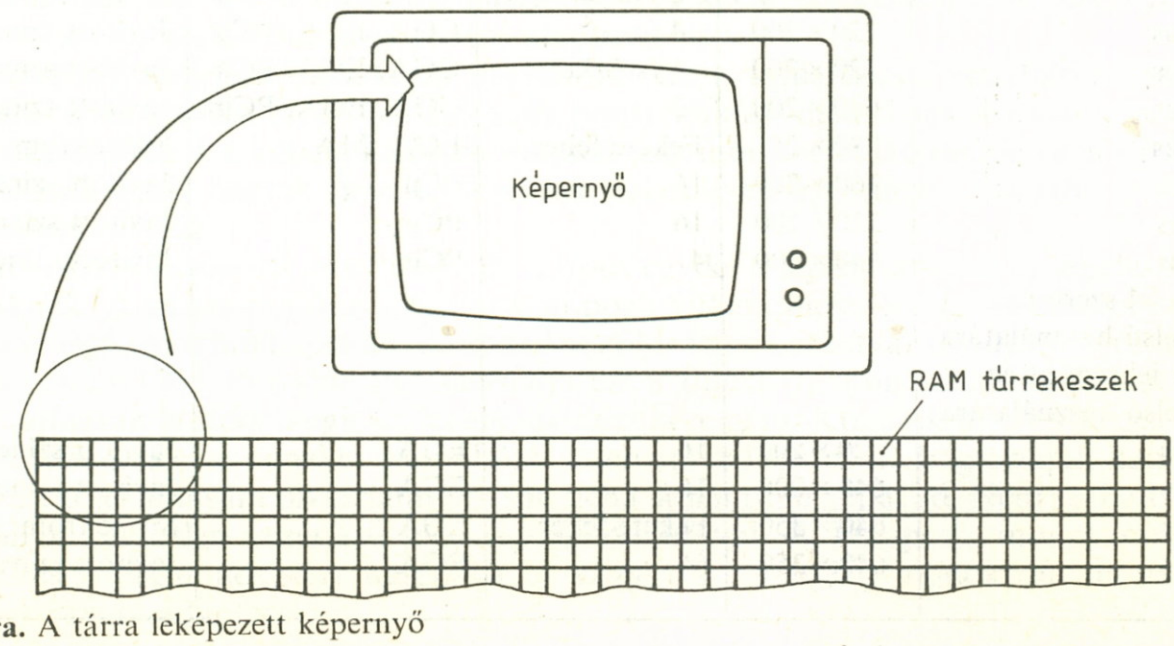
A képernyőtár fizikailag az adapter egyéb áramköreivel együtt az adapterkártyán található. Logikailag azonban (a CPU számára) a számítógép központi tárának részét képezi. Az A. és a B. tárblokk A0000 és BFFFF közötti 128K-ja a képernyőtár részére fenntartott terület. Az eredeti két képernyőadapter azonban ennek csak kis töredékét használja. A monokrom adapter 4K-t igényel a B000 című bekezdéstől. Az eredeti színes-grafikus adapter 16K-t használ a B800-as címtől. A fennmaradó rész, különösen az A000-tól B000-ig tartó 64K-s blokk, a fejlettebb megjelenítési módok számára van fenntartva; pl. ezt használja az EGA.



A PCjr eltér a családi hagyományoktól, mivel a tár alacsony című részét használja képernyő-memóriának. A gép speciális áramkörök segítségével nagyon hűen utánozza a színes-grafikus adaptert. Ez a VGA (Video Gate Array) nevű áramkör azt a látszatot tudja kelteni, mintha a PCjr képernyő-memóriája a színes-grafikus adapter B800-as címén lenne. A B800-as területre vonatkozó hivatkozásokat a VGA átirányítja a képernyőtár valódi címeire. A PCjr a RAM első 128K-jának tetszőleges részét használhatja videotárként; a VGA nyilvántartja a tényleges címet. Gyakorlatilag a PCjr-t úgy kezelhetjük, mint egy színes-grafikus adapterrel felszerelt PC-t, amiben a képernyőtár a B800 címen kezdődik.

A kép előállítása

A monokrom és a színes-grafikus adapter a képernyőn megjelenített adatokat tárra leképezve tárolja. Az elnevezés onnan ered, hogy a képernyőtár minden egyes címe a képernyő egy megadott helyének felel meg. (L. a 4.1. ábrát.) A megjelenítő áramkör ciklikusan kiolvassa a tár tartalmát, és a képernyőre küldi. Az információ olyan sebességgel változhat, amilyennel a programok adatokat tudnak írni a képernyő tárolójába. A CRT-vezérlő a kapcsolat a kép-



4.1. ábra. A tárra leképezett képernyő

ernyőtár és a képernyő között, a memóriából érkező bitáradatot a képernyő megfelelő pontjain fénypontokká alakítja át.

Ezeket a fénypontokat vagy fénycsíkokat *pixelnek* (képpontoknak) nevezik, és a képernyő foszforeszkáló felületére becsapódó elektronsugár váltja ki. Az elektronsugarat egy olyan elektronágyú bocsátja ki, amely soronként végigpásztázza a képernyőt. Miközben az ágyú vízszintesen és fentről lefelé (az ún. raszter mentén) végigpásztázza az ernyőt, a CRT-vezérlő olyan videojeleket állít elő, amely a tárolt biteknek megfelelően bekapcsolja, ill. kioltja az elektronsugarat.

A videoáramkör másodpercenként 60-szor frissíti fel a képet, így a változó képek tiszták és stabilak. Minden frissítő ciklus végén az elektronsugárnak a képernyő jobb alsó sarkából a bal felső sarokba kell visszatérnie, hogy az új ciklust elkezdje. Ezt *függőleges visszafutás*nak nevezzük. A visszafutás ideje alatt a sugarat ki kell oltani, és információ sem írható a képernyőre.

A függőleges visszafutás (1,25 ms) egy olyan ok miatt fontos a programozó számára, ami némi magyarázatot igényel. A képernyőadapter memóriájának különleges kétkapus tervezése egyforma hozzáférést biztosít a CRT-vezérlő és a CPU számára a képernyő-memóriához. Ezzel a CPU és a CRT-vezérlő egyidőben férhet a videomemóriához. Ha a CPU azalatt nyúl a tár egy bájtjához, mialatt a CRT-vezérlő a képernyőre ír, ez rövid ideig tartó „havazást" okoz a képernyőn. Ha viszont arra utasítjuk a CPU-t, hogy csak a függőleges visszafutás ideje alatt nyúljon a tárhoz, amikor a CRT-vezérlő nem foglalkozik a tárral; a havazás elkerülhető. A színes-grafikus adaptert használó rendszerekben a programok az adapter egyik I/O portján (hexa 3DA című) megvizsgálhatják a *függőleges szinkronjel* nevű állapotbitet. Ez a bit 1

4.1. táblázat
A 15 megjelenítési mód jellegzetességei

Mód	Típus	Méret	Színek	Adapter	Képernyő
0	Szöveges	40 × 25	16 (szürke)	CGA, EGA, PCjr	Javított színes
1	Szöveges	40 × 25	16 Előtér 8 Háttér	CGA, EGA, PCjr	Javított színes
2	Szöveges	80 × 25	16 (szürke)	CGA, EGA, PCjr	Javított színes
3	Szöveges	80 × 25	16 Előtér 8 Háttér	CGA, EGA, PCjr	Javított színes
4	Grafikus	320 × 200	4	CGA, EGA, PCjr	Javított színes
5	Grafikus	320 × 200	4 (szürke)	CGA, EGA, PCjr	Javított színes
6	Grafikus	640 × 200	2	CGA, EGA, PCjr	Javított színes
7	Szöveges	80 × 25	Fekete-fehér	EGA, MA	Monokrom
8	Grafikus	160 × 200	16	PCjr	Javított színes
9	Grafikus	320 × 200	16	PCjr	Javított színes
10	Grafikus	640 × 200	4	PCjr	Javított színes
11	Minden jel szerint az EGA belső használatára				
12	Minden jel szerint az EGA belső használatára				
13	Grafikus	320 × 200	16	EGA	Javított színes
14	Grafikus	640 × 200	16	EGA	Javított színes
15	Grafikus	640 × 350	Fekete-fehér	EGA	Monokrom
16	Grafikus	640 × 350	64	EGA	Javított színes

a függőleges visszafutás megkezdésekor és a végén. Ebben az 1,25 msec-os szünetben annyi adatot írhat a program a videomemóriába, amennyit csak tud. A visszafutás után a CRT-vezérlő havazás nélkül ki tudja írni ezeket az adatokat az ernyőre. Ez a technika minden olyan alkalmazásnál hasznos, ahol gyors egymásutánban tiszta ábrákat kell megjeleníteni.

Megjelenítési módok

Eredetileg az IBM személyi számítógépeken 8 megjelenítési formátum (más néven mód) volt. Ez később még legalább 7-tel bővült. A megjelenítési módok szabják meg a kép jellemzőit: a megjeleníthető szöveg mennyiségét, a grafika felbontását (részletességét) és a színeket. A színes-grafikus adapter számos formátummódot kínál mind a szöveges, mind a grafikus módokban. A monokrom adapter mindössze egyetlen egyszínű szöveges formát biztosít. A PCjr és az EGA-kártya a régi és új formátumok egész sorát nyújtja.

A megbeszélendő 15 módot egy 0 és 16 közötti szám azonosítja. (L. a 4.1. táblázatot.) A 0 és 3 közötti módok a színes-grafikus adapter szöveges módjai, a 4-es, 5-ös, 6-os módok pedig ugyanennek az adapternek a grafikus módjai. A 7-es mód monokrom szöveges formátum, ezt csak az IBM monokrom adapteren vagy ezzel egyenértékű berendezésen lehet használni. A 8-as, 9-es, 10-es módokat a PCjr-nél vezették be, itt használhatók a 0 és 6 közötti módok is, de ezek nem használhatók a szabványos IBM képernyőadaptereknél. A 13 és 16, ill. a 0 és 7 közötti módokat is az EGA használja.

Színek minden megjelenítési módban használhatók, kivéve a monokrom adapter számára fenntartott 7-es módot. A színes-grafikus adapternél rendelkezésre álló módokkal kettőtől tizenhatig terjedő színkombinációból, továbbá a szürke különböző árnyalataiból választhatunk (ez utóbbit színelnyomásos módnak nevezzük).

Monitorok

A használható képernyős berendezés (a monitor) típusa jelentős hatással van a programtervezésre. Sok monitor nem képes színt vagy grafikát előállítani, másoknak olyan rossz minőségű a képe, hogy csak a 40 oszlopos szöveges mód használható. A PC-számítógépcsalád sokféle típusú monitort használhat. A két fő kategória a monokrom és a színes monitor, és ezen belül négy fontos alaptípus van.

Közvetlen vezérlésű monokrom monitorok. Ez a típus képes nagyfelbontású szöveg és karakteres grafika megjelenítésére, de pontgrafikára nem. Csak a monokrom adapterrel működik. Azok a képek, amelyek minden más monitoron látszanak, ezen egyáltalán nem jelennek meg, hacsak nem használunk valamilyen különleges illesztőt, pl. a Hercules adapterkártyát.

Monokrom kompozit monitorok. Ez a típus egyike a legelterjedtebb és legolcsóbb monitoroknak. A színes-grafikus adapter kompozit videokimenetére kell csatlakoztatni. Elég jó minőségű egyszínű képet ad, rendszerint zöld- vagy borostyánszínűt. A hordozható Compaq és az IBM Portable PC használja ezt a típust. A monokrom kompozit monitorok alkalmasak grafika megjelenítésére, de színekére nem. Közülük néhány képes némi „színvisszaadásra", a színeket különböző intenzitással pótolva. Legtöbbjük azonban színes bemenőjel hatására olvashatatlan, sőt, néha láthatatlan képet ad. Nem szabad összetéveszteni a közvetlen vezérlésű monokrom monitorral. A kompozit monokrom monitor a színes-grafikus adaptert használja, a közvetlen vezérlésű monokrom monitor pedig a monokrom adaptert.

Kompozit színes monitorok és televíziókészülékek. A megjelenítő képét egyetlen összetett jel állítja elő, amely a színes-grafikus adapter kompozit videokimenetén át érkezik. Színek és grafika előállítására is alkalmas, de megvannak a korlátai. A 80 oszlopos kép gyakran olvashatatlan, csak bizonyos színkombinációk érvényesülnek igazán, ezenkívül a grafika felbontása gyenge minőségű –, így csak durva felbontású grafikai módok használhatók.

Bár a kereskedelemben kapható televíziókészülék (akár színes, akár fekete-fehér) technikai értelemben kompozit monitorok, képük rendszerint még gyengébb minőségű egy kifejezetten erre a célra készített kompozit monitorénál. Szöveg megjelenítésére a 40 oszlopos, esetleg a 20 oszlopos módot kell használni, csak így biztosítható a kép olvashatósága. A tv-t a színes-grafikus adapter kompozit videokimenetére kell csatlakoztatni (a PCjr-nek külön tv kimenete van), de a kompozit jelet egy RF-adapter segítségével át kell alakítani, mielőtt a tv-re csatlakoztatnánk.

RGB színes monitorok. A két világ legjobb tulajdonságait egyesítik. Rendelkeznek a monokrom monitor kiváló minőségű szövegmegjelenítő képességével, a nagy felbontású grafikával és a színekkel. Az RGB a red-green-blue (vörös, zöld, kék) rövidítése. Nevüket onnan kapták, hogy minden szín jeléhez külön vezetéket használnak – ellentétben a kompozit monitor egy vezetékével. A vezetékeket a színes-grafikus adapter RGB bemenetére kell csatlakoztatni. Egy csúcsminőségű RGB monitor képét tisztaságban, olvashatóságban csak az IBM monokrom monitorja múlja felül. A kép és a színek minősége rendszerint sokkal jobb annál, mint amit a kompozit videokimenetre csatlakozó bármilyen monitor előállít.

Bár a monokrom monitornál nem választhatunk színt a karakterek megjelenítésénél, rendelkezésre áll néhány olyan lehetőség, ami részben pótolja a színeket. Ezek a fényes és a normál intenzitás, az aláhúzás és az inverz kiírás. (☛ A színek használatát szöveges és grafikus módokban, valamint a monokrom adapter viselkedését színek alkalmazásakor „A színek használata" című részben beszéljük meg.)

A képernyő felbontása

A grafikus képek egyedi pontokból épülnek fel. Ezek a képelemek. A képernyő felbontását a képernyő sorainak számával, és az egy sorban található képpontok számával definiáljuk. A monitor sorainak számát a hardver és a videojel határozza meg, és erre szinte semmi befolyásunk nincs. A szabványos PC képernyőjére 25 szövegsor és 200 grafikus sor fér. A képernyő felbontását tehát az egyes sorokban látható képpontok számának megváltoztatásával érhetjük el.

A PC grafikájának három finomsági fokozata van: durva, közepes és finom. A képpontok száma az egyes fokozatokban 160, 320 és 640 soronként. [JR] A durva fokozatot a PCjr-nél vezették be. Az eredeti színes-grafikus adapter, valamint az EGA-kártya nem támogatja. Mivel szöveges karakterek grafikus módokban is megjeleníthetők, ebből következik, hogy a közepes és a finom grafikában a szöveg mérete azonos. (L. a 4.2. táblázatot.)

4.2. táblázat
A grafikus módokban kirajzolt karakterek felbontása

Felbontás	Képpontok	Karakterek
Durva	160 × 200	20 × 25
Közepes	320 × 200	40 × 25
Finom	640 × 200	80 × 25

A 80 oszlopos, 25 soros formátumnál használt keskenyebb karakterek 640 × 200-as felbontást használnak, a 40 oszlopos, 25 soros formátum karakterei pedig 320 × 200-as felbontást. (A monokrom adapter 80 × 25 karakteres formátumának azért olvashatóbb a szövege, mert a karaktereket nagyobb felbontású képpontokkal – 720 × 350 – jeleníti meg.)

Később látni fogjuk, hogy a durva felbontású grafikánál van egy saját, 20 oszlopos szövegformátum, amelynek nincs megfelelője a szabványos szöveges formátumok között. JR A 20 oszlopos szöveget (a hozzátartozó durva felbontású grafikával együtt) a PCjr-nél vezették be, más adaptereknél ez a formátum hiányzik.

A megjelenítési módok vezérlése

A megjelenítési módokat a ROM-BIOS 16-os megszakításának 0-ás szolgáltatásával szabályozzuk. (☛ L. a 9. fejezetet.) A BASIC **SCREEN**-utasításával teljes mértékben szabályozhatjuk a megjelenítési módokat, de a BASIC saját szisztémát használ: a módok száma eltér a

4.3. táblázat
A megjelenítési módok megváltoztatására használt BASIC- és DOS-utasítások

Mód	Módot váltó BASIC-utasítás	Módot váltó DOS-utasítás
0	SCREEN 0,0 : WIDTH 40	MODE EW40
1	SCREEN 0,1 : WIDTH 40	MODE CO40
2	SCREEN 0,0 : WIDTH 80	MODE BW80
3	SCREEN 0,1 : WIDTH 80	MODE CO80
4	SCREEN 1,0 vagy SCREEN 4	n/a
5	SCREEN 1,1	n/a
6	SCREEN 2	n/a
7	n/a	MODE MONO
8	SCREEN 3	n/a
9	SCREEN 5	n/a
10	SCREEN 6	n/a

ROM-BIOS által használttól. A megjelenítési módok némelyikét a DOS segítségével is szabályozhatjuk. A DOS-parancs szintjén azonban csak szöveges módokat lehet beállítani, nincs olyan DOS-parancs, amellyel grafikus módba lehetne átkapcsolni. Ez látható a 4.3. táblázaton.

A színek használata

A monokrom adapternél használt megjelenítési mód kivételével az összes többi módban nagy a színválaszték. Már fölfigyelhettünk arra, hogy a különböző módokban különböző számú szín áll rendelkezésünkre. Ebben a részben ismertetjük a megjelenítési módok színlehetőségeit.

A PC képernyőjén megjelenő színek négy elem kombinációjaként jönnek létre: három színből vörös, zöld, kék; és egy intenzitás- vagy fényességkomponensből. A szöveges és a

grafikus módok ugyanazokat a szín- és intenzitáslehetőségeket használják, de eltérő módon kombinálják ezeket a színes kép előállításához. A szöveges módok – amelyeknek alapeleme a több képpontból felépülő karakter – egy teljes bájtot használnak fel a színek, az intenzitás, valamint a karakter, ill. a karakterháttér esetleges villogásának beállítására. A grafikus módok alapeleme sokkal kisebb (egy képpont), és ezek 1–4. bitet használják a szín és az intenzitás beállítására. A képpont nem villoghat és nincs háttere. ☛ A szöveges és grafikus módok jellemzőinek beállításával a következőkben külön is foglalkozunk. Először szóljunk magukról a színekről.

A BASIC által használt színszámokat – ezek a 0 és 15 közötti számok azonosítják a PC színeit – úgy származtathatjuk, hogy a szín négy alkotóelemét egy bináris szám bitjeinek képzeljük. (A színeket és bitekre bontott kódjaikat a 4.4. táblázatban láthatjuk.) Ha 16 színű

4.4. táblázat
A PC-család teljes színskálája, az egyes színeket megadó 4 bittel együtt

Intenzitás	Vörös	Zöld	Kék	Szám	Jelentése
0	0	0	0	0	Fekete
0	0	0	1	1	Kék
0	0	1	0	2	Zöld
0	0	1	1	3	Ciánkék
0	1	0	0	4	Vörös
0	1	0	1	5	Bíborvörös
0	1	1	0	6	Barna (vagy sötétsárga)
0	1	1	1	7	Világosszürke (vagy közönséges fehér)
1	0	0	0	8	Sötétszürke (sok képernyőn fekete)
1	0	0	1	9	Világoskék
1	0	1	0	10	Világoszöld
1	0	1	1	11	Világos ciánkék
1	1	0	0	12	Világosvörös
1	1	0	1	13	Világos bíborvörös
1	1	1	0	14	Sárga (vagy világossárga)
1	1	1	1	15	Vakítófehér

módot használunk, akkor megkapjuk az összes színt 0 és 15 között, ha 8 színűt, akkor a 0 és 7 közötti színeket kapjuk; vagyis az összes színt fényes intenzitás nélkül. Egy 4 színű módnál 4 színt választhatunk a 16 elemű listából. A kiválasztott 4 színt *palettának* nevezzük. Egy 2 színű módnál a 0-s és a 7-es színt kapjuk – ez a fekete és a közönséges fehér.

Eddig a szabványos PC 16 színű palettájáról volt szó, ez tehát a vörös, a zöld és a kék színből, valamint az intenzitásból (jele I) áll. Ezt az alapvető sémát nevezhetjük IRGB-nek. A PC-család rendelkezik egy 64 színű palettával is, de ez csak az EGA-kártyával és a javított színes képernyő (ECD, Enhanced Color Display) együttesével használható. Ez a 64 színű paletta a szokásos három színre épül, de minden egyes színhez két független jel tartozik: egy fényesebb és egy tompább változat. A 64 színű paletta jelzése RrGgBb – itt a nagybetűk a tompább színeket jelzik. Figyeljük meg, hogy nem két különböző intenzitási szintről van szó, hanem két külön színjelről, és ez mind a három színnel négy intenzitást tesz lehetővé. A vörös

színnél pl. a négy intenzitás a következő: Rr (a legintenzívebb), R., .r és .. (nincs vörös). Az RrGgBb összes lehetséges kombinációja adja a 64 színt.

Az EGA/ECD kombináció 64 színű palettájának részleteiben nem mélyülünk el, mivel elég ritka és speciális, ezenkívül nem is esik igazán egybe a PC-család fő vonalával. (Ha valóban az lenne a célunk, hogy elmélyedjünk az összes különleges, a fő vonaltól távol eső kiegészítő elemben, akkor ennek a könyvnek a sokszorosát is megtölthetnénk vele.) Egy ennél is ritkább adapter – képernyő kombináció az IBM professzionális grafikus adapterének és képernyőjének 256 színe és rendkívül nagy a felbontása. Ez azonban még távolabb esik a PC fő vonalától, tehát ennek az ismertetését is elhagyjuk, és visszatérünk a szabványos paletták részletesebb ismertetésére.

A színek kiválasztásakor több dologra is figyelni kell. A szín négy alkotóeleme (IRGB) aktívan részt vesz a szín létrehozásában. Minél több elem aktív, annál fényesebb a szín, mégis „kimosódottabb" benyomást kelt. A szemnek az egyedi, tiszta színek – vörös, zöld, kék – vizuálisan intenzívebbeknek tűnnek, mint akár a kevert színek (ciánkék, bíborvörös, sárga), akár a tiszta színek ún. intenzívebb (fényesebb) változata. Érdemes figyelembe venni a színek kiválasztásakor a következőket is:

- Egyes színes képernyők nem reagálnak az intenzitásbitre. E hiányosságnak a következménye, hogy a 8-as szám azonos lesz a 0-ással, a 9-es az 1-essel (és így tovább).
- Ha kompozit monokrom képernyőt használunk színes-grafikus adapterrel, akkor a feketén (0) és a fehéren (7) kívül a többi szín olvashatatlan.
- Végül a PC-n vagy XT-n futó programoknál (az IBM monokrom adapter használatakor) figyelembe kell venni, hogy a monokrom képernyő furcsán kezeli a színeket. (☛ L. az attribútumok beállítását.)

A színek megválasztásakor ellenőrizzük a következő részekben leírtakat is. Mindegyikben találunk a színekről egy-egy fontos megjegyzést.

Színelnyomásos módok

Annak érdekében, hogy a grafikus módok a lehető legtöbb színes és monokrom képernyőn kompatíbilisek legyenek, az IBM bevezetett néhány olyan módot is, amelyik nem ad színeket. Ezeket nevezzük színelnyomásos módoknak. Három ilyen mód van: a 0-s, a 2-es és az 5-ös. Ezeknél a módoknál a színek a szürke különböző árnyalataivá alakulnak át, ill. annak a

4.5. táblázat
A 0–5-ös módok színparamétere. A 0–3-as és a 4–5-ös más rendszerű!

Mód	Színelnyomásos	Színes
0	SCREEN 0,0 : WIDTH 40	
1		SCREEN 0,1
2	SCREEN 0,0 : WIDTH 80	
3		SCREEN 0,1 : WIDTH 80
4		SCREEN 1,0
5	SCREEN 1,1	

színnek az árnyalataivá, amit az ernyő foszforeszkáló bevonata létrehoz. Az 5-ös módban négy árnyalat van, a 0-s és a 2-es módban viszont széles a választék. A színek az adapter kompozit kimenetén elnyomva jelennek meg, az RGB-kimeneten *nem*. Ez az inkonzisztencia egy elkerülhetetlen technikai korlát eredménye.

MEGJEGYZÉS: minden színelnyomásos módnak van egy színes megfelelője; a 0-s és 1-es mód 40 oszlopos szöveges, a 2-es és 3-as mód 80 oszlopos szöveges, a 4-es és 5-ös mód közepes felbontású grafikát jelent. Az a tény, hogy a 4-es és 5-ös módnál megfordul a színes és színelnyomásos módok sorrendje, a BASIC-ben bizonyos bonyodalmakat okozott. A BASIC **SCREEN** utasításának paramétere szabályozza a színt. Ennek a paraméternek az értelme megfordul a 4-es és 5-ös módoknál; vagyis a szöveges módoknál **SCREEN,1** kapcsolja be a színt, a grafikus módoknál (4-es és 5-ös) viszont elnyomja. Ezt az inkonzisztenciát eleinte tekinthették programozási hibának, ma azonban már a **SCREEN**-utasítás hivatalos definíciójának része. A 4.5. táblázatban láthatók a **SCREEN** utasítás helyes formái a 0 és 5 közötti módokra.

Színek a szöveges és a grafikus módoknál

Tudnunk kell, hogy a szöveges és a grafikus módokban a színek használata eltér. Különösen fontos ismerni azt a nyilvánvaló inkonzisztenciát, ami a szöveg színeinek használatakor jelenik meg. Szöveges módban minden egyes karakterpozíció színét a többitől teljesen függetlenül állíthatjuk be: szabadon használhatjuk a teljes 16 színű palettát az előtérben, és a 8 színű palettát a háttérben. Grafikus módoknál minden egyes képpont színét, ill. a grafikus rajzoló utasítások (pl. a BASIC-ben lévők) színét szabhatjuk meg.

Elméletben a grafikus módoknak a színek gazdagabb használatát kellene nyújtani a teljes képernyőn. Figyelembe kell azonban venni, hogy ha grafikus módban szöveget írunk, akkor a háttér színét nem befolyásolhatjuk karakterenként: mindig az egységesen beállított háttérszín érvényesül. (☛ L. a 4 színű mód 0-s palettájának ismertetését.) Ez azt jelenti, hogy elvileg a grafikus mód nagyobb beleszólást enged a színekbe, ha viszont szöveget írunk ki, gyakorlatilag mégis kevesebb a beleszólásunk. Ez a grafikus mód szövegkiírási szolgáltatásának eredendő jellemzője. (☛ L. a 9. fejezetet.)

Színek beállítása a szöveges módokban

Szöveges módokban a képernyő minden egyes karakterének pozícióját a tár két szomszédos bájtja vezérli (☛ A 3. fejezetben foglalkoztunk bővebben ennek a két bájtnak a helyével.) Az 1. bájt tartalmazza a kiírandó karakter ASCII kódját. (☛ A *C* függelék tartalmazza a karakterek táblázatát.) A 2. bájt szabja meg a karakter megjelenítését: a színét és a többi jellemzőt. A 2. bájtot nevezzük a karakter attribútumának.

Mielőtt tovább haladnánk, meg kell magyarázni néhány fogalmat. Az IBM-nél a képernyővel kapcsolatos szóhasználatban a *szín* és az *attribútum* szinonímák. Bár a két fogalomnak pontos technikai jelentése van (ezek rokonok, de némileg eltérnek egymástól), gyakran fogunk találkozni a két fogalom pontatlan használatával. Ebben az esetben mindkettőn majdnem ugyanazt értik! A zavar elkerülése végett állapodjunk meg abban, hogy a két fogalmon (igaz, kissé pongyolán) a karakter megjelenítését, valamint a megjelenést megszabó tárbeli kódolást is értjük.

A szöveges karakter attribútumának három összetevője van: az előtér színe (magának a karakternek a színe), a háttér színe (a karakter által el nem takart terület színe) és a karakter

villogása. Az előtér a teljes PC-családban 16 színű lehet. A háttér színét 8 színből választhatjuk ki: ezek a 0 – 7-es színek, azaz az alapvető színek, intenzitás nélkül.

A képernyő összes karakterének külön lehet szabályozni az attribútumait, teljesen függetlenül a képernyő többi karakterétől. Az attribútum bájt 8 bitje segítségével mindentől függetlenül beállíthatjuk a képernyő attribútumának egy elemét. (A bitek beállítása a 4.6. táblázatban látható.) A DOS és a BASIC alapállásban a hexa 07 értéket használja, ez normális fehér (7) fekete (0) háttér előtt, villogás nélkül.

4.6. táblázat
A színattribútum bájtjainak kódolása

A bit helye 7 6 5 4 3 2 1 0	Jelentése
1	Az előtérkarakter villogása
. 1	A háttérszín vörös komponense
. . 1	A háttérszín zöld komponense
. . . 1	A háttérszín kék komponense
. . . . 1 . . .	Az előtérszín intenzitás komponense
. 1 . .	Az előtérszín vörös komponense
. 1 .	Az előtérszín zöld komponense
. 1	Az előtérszín kék komponense

A képernyő összes karakterének külön lehet szabályozni az attribútumait, teljesen függetlenül a képernyő többi karakterétől. Az attribútum bájt 8 bitje segítségével mindentől függetlenül beállíthatjuk a képernyő attribútumának egy elemét. (A bitek beállítása a 4.6. táblázatban látható.) A DOS és a BASIC alapállásban a hexa 07 értéket használja, ez normális fehér (7) fekete (0) háttér előtt, villogás nélkül.

JR

JR Ha a PCjr-nél 16 színű grafikus módban (8-as és 9-es mód) szöveget írunk ki, akkor mind a 16 színt használhatjuk háttérként, de a kiválasztott háttérszín az összes karakterre érvényes. Bár az első pillanatban erős korlátozásnak tűnik, valójában igen hasznos lehet. Elérhető pl., hogy láthatatlan karaktereket írjunk a képernyőre, ha az előtér és a háttér színét azonosnak választjuk. Ez eszményi módszer arra, hogy jelszavakat és más bizalmas adatokat láthatatlanul be lehessen írni. A monokrom adapternél is van egy ilyen láthatatlan mód. (☞ L. attribútumok beállítása.)

A színek minősége függ a monitortól. Sok színes képernyőnél a fényes, nagy intenzitású szín jól olvasható az azonos színű, de alacsony intenzitású háttéren. Másrészt sok színes képernyő nem reagál az intenzitás változására. Ezeknél a képernyőknél gyakran az egyébként olvasható előtér háttér színkombinációk – pl. sárga barna háttér előtt – megkülönböztethetetlenek.

Bár a grafikus módokban is szépen lehet szöveget kiíratni, több nyomós érv szól amellett, hogy a szöveget szöveges módban írassuk ki. A gyorsaság a legfőbb előny, ha szöveges módban írjuk ki a karaktereket. Ez részben annak tulajdonítható, hogy a szöveges módú karaktereket egy karaktertáblázatból veszi a gép; a grafikus módú karaktereket pedig bitenként kell kirajzolni a tárból. A szöveges mód kevesebb tárat használ fel, mint a grafikus mód, ezzel több memória marad a megjelenítendő oldalak számára. Ez lehetővé teszi, hogy több „oldalnyi" szöveget tároljunk közvetlenül a képernyőtárban, és onnan gyors egymásutánban meg is

jeleníthetjük őket. Ezenkívül szöveges módú karaktereknél több lehetőség áll rendelkezésre. Egyrészt nagyobb a színválaszték, másrészt szöveges módban a karakterek villogtathatók, míg a grafikus módokban egyáltalán nincs villogtatási lehetőség.

Attribútumok beállítása a monokrom módban

Az IBM monokrom adapter által használt 7-es mód – ún. monokrom mód – biztosít néhány, a színeknek megfelelő megjelenítési módot. A monokrom karakterek megjelenítési attribútumait azzal az általános kódolási sémával tudjuk beállítani, amivel a 0–3-as grafikus módok szövegkaraktereit.

A monokrom módnál az intenzitás- és a villogásbiteket használjuk. Az előtér–háttér „színkombinációk" közül azonban csak négynek van megkülönböztethető hatása:

- Normális fehér karakter, fekete háttér előtt. Ezt a fehér előtérbitek (111) és a fekete háttérbitek (000) kiválasztásával kapjuk. A végső érték hexa 07.
- Aláhúzott karakterek. Ez a hexa 01 értékű attribútumbájt, ami tulajdonképpen kék előtér (001), fehér háttéren (000).
- Inverz, vagyis fekete karakter (000), fehér háttér (111) előtt. A végső érték hexa 70.
- Láthatatlan karakterek. Ezeket úgy kapjuk, hogy fekete karaktert (000) írunk fekete (000) háttérre. A végeredmény hexa 00.

A többi színkombináció eredménye azonos azzal, mintha közönséges fehérrel írnánk feketére vagyis hexa 07. Más – egyébként logikusnak tűnő – színkombinációk (pl. láthatatlan fehér fehéren vagy inverz aláhúzva) nem léteznek a monokrom módban. Csak a fent leírt négy. Figyeljük meg, hogy a villogást és az intenzitást szabályozó bit független ettől a négy színkombinációtól.

A szín beállítása grafikus módokban

Az eddigiekben azt láttuk, hogyan lehet a színeket beállítani a monokrom képernyő szöveges módjában. Grafikus módokban a színek beállítása egészen más. A grafikus módokban (a 4-es és a 6-os, ill. a 8-as és a 10-es közötti módok) minden képponthoz tartozik szín. A szín beállítása azonos az attribútumok beállításával szöveges módban, de vannak fontos eltérések. Először: a képpontok nem villoghatnak. Másodszor: minden képpont külön színes pont (nincs előtér és háttér), ezért aztán minden képpontnak van valamilyen színe. Amikor grafikus módban szöveget írunk, egy színt használunk a hátteret alkotó képpontokhoz, egy tetszőleges színt pedig a karaktert alkotó képpontokhoz.

🕮 MEGJEGYZÉS: Ha BASIC-ban használjuk a grafikus módot, az a benyomásunk támadhat, hogy a grafikus módban van háttérszín. Itt azonban csak a BASIC által alkalmazott célszerű konvencióról van szó: minden képpont, amit nem állítunk be explicit módon valamilyen előtérszínre, a háttér színét kapja. A ROM-BIOS videoszolgáltatásra szintén alkalmazzák ezt a háttérszín konvenciót. (☛ L. a 9. fejezetet.)

Minden grafikus módnál megtaláljuk a színek előre definiált választékát, ezt palettának nevezzük. A szabványos paletta megváltoztatható a PCjr-nél és az EGA-nál, de az eredeti színes-grafikus adapternél nem. Ha egy grafikus módban kiválasztottuk a paletta színeit, az egyes képpontok színét úgy választhatjuk a rendelkezésre álló színekből, hogy beállítjuk a képponthoz tartozó bitek színértékét. Egy 2 színű módban minden képponthoz egy bit tartozik, és a képpont színértéke 0 vagy 1 lehet. Egy 4 színű módban 2 bit van, és a színek értéke 0 és

4.7. táblázat
A kétszínű grafikus mód – a 6-os mód – szabványos palettája

Bit	Érték	Szín
0 0	0	Fekete
0 1	1	Fehér

3 között lehet. Egy 16 színű módban 4 bit van, és a színek értéke 0 és 15 között lehet. A képpontot definiáló színérték nem szükségszerűen azonos a képernyőn megjelenő szín tényleges számával.

A 6-os számú 2 színű módnál csak egy szabványos paletta van, ez látható a 4.7. táblázatban. A 4 színű 4-es és 5-ös módoknál két szabványos paletta van. (A 0-s a 4.8. táblázatban, az 1-es pedig a 4.9. táblázatban látható.) A palettákkal kapcsolatban két dolgot kell megemlíteni. Először: a paletta 0-s színe feketéről (0-s szín) tetszőleges színre változtatható. Másodszor: a paletta 0-s színe a háttér színe, a 3-as színe pedig az előtér színe, ha szöveget írunk ki. A 4 színű 10-es módban egy szabványos paletta van, ez azonos az 1-es palettával. A 16 színű 8-as, 9-es, 13-as és 14-es módokban egy szabványos paletta van. Ezen a palettán – a várakozásnak megfelelően – a 0 és 15 közötti értékek megegyeznek a színek tényleges számaival. Ne feledkezzünk meg arról, hogy a 8-as és 9-es mód csak a PCjr-nél, a 13-as és 14-es mód pedig csak az EGA-nál használható.

4.8. táblázat
A négyszínű grafikus módok – a 4-es és az 5-ös mód – két szabványos palettájának egyike: a 0-ás paletta

Bit	Értéke	Szín
0 0	0	Fekete (kiindulási érték; tetszőleges színre változtatható)
0 1	1	Zöld
1 0	2	Vörös
1 1	3	Barna

4.9. táblázat
A négyszínű grafikus módok – a 4-es és az 5-ös mód – két szabványos palettájának egyike: az 1-es paletta

Bit	Értéke	Szín
0 0	0	Fekete (kiindulási érték; tetszőleges színre változtatható)
0 1	1	Ciánkék
1 0	2	Bíborvörös
1 1	3	Közönséges fehér

A paletták színeinek újradefíniálása a PCjr-nél és az EGA-nál

Egészen eddig a szabványos paletták használatával létrehozható szabványos színekről volt szó. Az eredeti színes-grafikus adapternél a paletták színkiosztása rögzített, nem változtatható meg. A PCjr-nél – és minden olyan képernyő-adapternél, amelyikbe betervezték ezt a szolgáltatást (pl. az EGA-nál) – a paletták színei megváltoztathatók. A színek újra defíniálása pusztán a színérték megváltoztatását jelenti, tehát pl. az 1-es szín a kéket, a 4-es szín pedig a vöröst jelentheti.

Egy kért palettaérték és a tényleges szín megfeleltetése az éppen használt paletta irányítása alatt áll. A palettát válthatjuk a BASIC paletta utasításával vagy a BIOS videoszolgáltatásaival. (☛ L. a 9. fejezetet.)

A képernyő-memória belseje

Végre elérkeztünk a képernyőtérkép belső működéséhez. Ebben a részben megismerjük, hogy kapcsolódik a képernyőtárban tárolt információ a képernyőhöz.

Jó tisztában lenni azzal, hogy a B. blokkot használó megjelenítési módoknál – a 0 és 6 közötti színes-grafikus és a 7-es monokrom módnál – programjainkkal bátran keresgélhetünk a képernyőtárban. Ez még a PCjr esetében is így igaz, bár az a *látszat,* hogy a B. blokkot használja a 0 – 6-os módokba. Eredetileg az IBM szerette volna elkerülni, hogy a programok közvetlenül hozzányúljanak a képernyőtárhoz, de mivel a legtöbb valamirevaló program ezt teszi, az IBM belenyugodott. Ma már arra törekszik, hogy az összes jelenlegi és jövőbeni képernyőadapternél támogassa ezt. Ugyanakkor az IBM ezeknél a módoknál meg is húzza a határt. Az újabb, fejlettebb módoknál – pl. a PCjr 8-as, 9-es, 10-es módjainál és az EGA 13 – 16-os módjainál – az IBM mindent elkövet, hogy elvegye mindenki kedvét a képernyőtár változtatgatásától. Az EGA-nál a képernyőtár elvileg az A. blokkban van, de programjaink számára elérhetetlen ezen a címen. Csak a bolondok próbálják meg áttörni ezt a korlátot.

A képernyőtár használata és kódolása attól függően más és más, hogy a 0 és 10 közötti módokból melyiket használjuk. (Emlékeztetünk rá, hogy a 0 – 6-os módok az eredeti IBM színes-grafikus adapterhez tartoznak, a 7-es mód pedig az IBM monokrom adapterhez. A 8 – 10-es módokat a PCjr-hez vezették be, és ez a gép a 0 – 6-os módokat is használja; ezek a különleges módok nem használhatók a színes-grafikus adapternél. A 11 – 16-os módok csak az EGA-nál használhatók.)

A 0 – 6-os módoknál és a 8-as módnál a képernyőtérkép 16K-t foglal el; a 9-es és 10-es módokban pedig 32K-t. A monokrom adapter 7-es módjánál csak 4K-t használ. Mind a monokrom, mind a grafikus adapternél a szöveges módok kevesebb helyet foglalnak el a grafikus módoknál, mert egy karakter ábrázolásához csak 2 bájt szükséges. (☛ Erről még részletesebben is szólunk.) Következésképpen egy 80-szor 25 karakteres ernyőkép mindössze 4000 bájtot igényel. Ezzel szemben egy grafikus kép terjedelme 16K és 32K között van, a használt színek számától függően. A 2 színű grafikus módokban egy képpont 1 bitet foglal el. A 4, ill. 16 színű módokban egy képponthoz (a nagyobb színszámok tárolása miatt) 2, ill. 4 bit tartozik. Ebből következik, hogy egy 320 × 200 képpontból álló, 16 színű, bitekre leképezett képernyő 32K-t igényel, tehát két képpont jut egy bájtra.

Mivel egy tipikus szöveges képernyő 4000 bájtot foglal el (40 oszlopos módban csak 2000 bájtot), marad némi hely szabadon a színes-grafikus adapter 16K-s képernyőtárból. Ezt a területet további szöveg számára hasznosíthatjuk, ha képernyőoldalakra osztjuk fel.

4.10. táblázat
Az egyes megjelenési módok által igényelt tárterület nagysága és kezdőcíme

Minimális tár		A kezdő bekezdés címe, hexa	Adapter
Mód	Felhasznált, K		
0	2	B800 (A cím más a PCjr-en)	CGA
1	2	B800 (A cím más a PCjr-en)	CGA
2	4	B800 (A cím más a PCjr-en)	CGA
3	4	B800 (A cím más a PCjr-en)	CGA
4	16	B800 (A cím más a PCjr-en)	CGA
5	16	B800 (A cím más a PCjr-en)	CGA
6	16	B800 (A cím más a PCjr-en)	CGA
7	4	B800	MA
8	16	PCjr fő tára (A cím változik)	n/a
9	32	PCjr fő tára (A cím változik)	n/a
10	32	PCjr fő tára (A cím változik)	n/a
13	32	A800	EGA
14	32	A800	EGA
15	64	A000	EGA
16	32	A800	EGA

Képernyőoldalak szöveges módoknál

A 0 – 3-as módoknál egyidőben 16K-nál kevesebb hely foglalt. A 0-s és 1-s mód 2K-t, a 2-es és 3-as mód 4K-t használ fel. Ezeknél a módoknál a rendelkezésre álló 16K tárat több képre, ún. *oldalakra* osztjuk fel. Egyszerre egy oldal (az aktív oldal) látszik. Információ írható akár a látható, akár a nem látható oldalakra. Ezzel a módszerrel összeállíthatjuk egy nem látható oldal tartalmát, mialatt más kép látszik, majd amikor eljön az ideje, az új képre válthatunk. Ha ilyen módon váltogatjuk a képeket, a váltás villanásszerűnek tűnik.

4.11. táblázat
A képernyőoldalak relatív kezdőcíme a 0 – 3-as módokban

Oldal	A 0-ás és az 1-es módban 2 K eltolás	A 2-es és a 3-as módban 4 K eltolás
0	B800	B800
1	B880	B900
2	B900	BA00
3	B980	BB00
4	BA00	
5	BA80	
6	BB00	
7	BB80	

A képoldalak számozása a 0-s és 1-es módban, 0-tól 7-ig tart, a 2-es és 3-as módban pedig 0-tól 3-ig; a 0-s oldal mindig a 16K-s képernyőtár elején kezdődik. Minden oldal egész K számú tárhatáron kezdődik. (A képernyőoldalak relatív kezdőcíme a 4.11. táblázatban látható.)

A látható oldalt úgy választjuk ki, hogy megváltoztatjuk a 6845-ös vezérlő IC-ben a kezdőcímet. Ezt rendszert a 16-os megszakítás 5-ös ROM-BIOS szolgáltatásával érjük el. (☛ L. a 9. fejezetet.)

A fenti módok bármelyikében megtehetjük, hogy az éppen nem látható oldalt egészen más célra használjuk fel, de ez nem szokás, és nem is tanácsoljuk. Ha ezt a potenciálisan szabad tárat más célra használjuk, ez csak az első lépés, következményei még sok bonyodalmat okozhatnak.

Képernyőoldalak grafikus módoknál

A PCjr-nél, az EGA-nál és minden olyan adapternél, ahol megvan a megfelelő nagyságú tár, a lap fogalma éppen olyan természetesen használható a grafikus, mint a szöveges módokban. Miért is ne lennének grafikus oldalak, ha egyszer a hozzájuk szükséges tár is megvan.

Több oldal használatának az a fő előnye szöveges és grafikus módban egyaránt, hogy lehetővé teszi a képek villanásszerű váltását anélkül, hogy az üres képernyőből kiindulva kellene a képet létrehozni. Elméletileg a grafikus módokban több oldalt fel lehetne használni finom átmenetek és részletdús animációs hatások létrehozására, de a rendelkezésre álló tár mennyisége ezt erősen korlátozza. Az újabb adaptereknél azonban több oldal is használható grafikai célokra.

Karakterek megjelenítése szöveges és grafikus módokban

Amint már láttuk, a monokrom és a színes-grafikus adapter szöveges módjaiban a memóriában nem a karakter képe, hanem csak a karakter ASCII kódja és attribútumai vannak. A karaktert az adapterben található karaktergenerátor rajzolja a képernyőre. A színes-grafikus adapter karaktergenerátora 8 × 8 képpontból építi fel a karaktert, a monokrom adapteré 9 × 14 képpontos formát használ. A nagyobb forma az egyik oka annak, hogy a monokrom adapter képe jobban olvasható.

A szabványos ASCII-karakterek – **CHR$(1) – CHR$(127)** – a használható ASCII-karakterkészletnek csak az egyik felét jelentik. Ugyanennek a karaktergenerátornak a segítségével még 128 grafikus karaktert is kiírhatunk – **CHR$(128)** és **CHR$(255)** között. Ezeknek közel a fele alkalmas arra, hogy egyszerű vonalas rajzokat készítsünk. (☛ A szabványos ASCII-karakterek és az IBM grafikus karaktereinek teljes listája a *C* függelékben található.)

Karakterek kiírhatók grafikus módban is, de ezek előállítása az előzőtől teljesen eltér. A grafikus módok csak bitenként tárolhatják az információt, és ez alól a karakterek sem jelentenek kivételt; őket is bitről bitre kell megrajzolni. Karakterek tekintetében az a bitekre leképezett képernyő előnye, hogy magunk is megtervezhetjük a karaktereinket. Az eredeti IBM monokrom adapterben a második 128 karakter táblázata RAM-ban van, ezért megváltoztatható. Ha a táblázatot megváltoztatjuk, akkor az IBM szabványos karakterkészlete helyett a saját tervezésű karaktereket használhatjuk és írhatjuk ki. A PCjr-nél mind a 256 karakter RAM-ban van, tehát az összes megváltoztatható.

A karakterek leképezése szöveges módokban

Szöveges módokban a tár térképe a képernyő bal felső sarkával kezdődik, és képernyő-pozícióként 2 bájtot használ föl. Az egymást követő karakterek abban a sorrendben találhatók a tárban, ahogy olvasni szoktuk őket, tehát balról jobbra és fentről lefelé.

A 0-s és 1-es szöveg módok 40 oszloposak és 25 sorosak. Helyfoglalásuk soronként $40 \times 2 = 80$ bájt. Egy képernyő csak 2K-t foglal el a 0-s és 1-es módban, így a 16K-s tár 8 képernyőoldalt tárolhat. Ha a sorokat 0-tól 24-ig, az oszlopokat pedig 0-tól 39-ig számozzuk, akkor egy tetszőleges karakter relatív tárcímét az első képernyőoldalon a következő BASIC-formula adja meg:

KARAKTER.OFFSET = (SOR.SZAM*80) + (OSZLOP.SZAM*2).

Mivel a karakter attribútumbájtja közvetlenül a karakter után található a tárban, a címét egyszerűen úgy kapjuk meg, hogy a karakter címéhez 1-et hozzáadunk.

A 2-es, 3-as és 7-es módok 40 helyett 80 oszlopos szövegmódok. A bájtok elhelyezése azonos, de az egyes sorokhoz kétszer annyi – $80 \times 2 = 160$ – bájt kell. Ennek következtében a 80-szor 25-ös képernyőforma 4K-t foglal el, és a 16K-s tárban négy oldal fér el. Az első oldal tetszőleges karakterének relatív címét a következő BASIC-képlet adja meg:

KARAKTER.OFFSET = (SOR.SZAM*60) + (OSZLOP.SZAM*2)

Ha a színes-grafikus adaptert használjuk, akkor minden oldal hagyományosan egész K határon kezdődik. Mivel szöveges módban egy képernyő 2000 vagy 4000 bájtot foglal el, a következő oldalig marad néhány fel nem használt bájt: 48 vagy 96, a sor hosszától függően. Ezért az általános formula (amelyik tetszőleges oldalon tetszőleges helyen álló karakter pozícióját megadja) a következő:

CIM = (SZEGMENS.BEKEZDES*16) + (OLDAL.SZAM*OLDAL.MERET)
+ (SOR.SZAM*SOR.SZELESSEG*2) + (OSZLOP.SZAM*2) + MELYIK

Az egyes jelölések:
CIM: a képernyőn tárolt információ 20 bites címe
SZEGMENS.BEKEZDES: a képernyő-memória címe (pl. B000 vagy B800)
OLDAL.SZAM: értéke 0 – 3 vagy 0 – 7 lehet
OLDAL.MERET: 2K vagy 4K
SOR.SZAM: 0 – 24 közötti érték
SOR.SZELESSEG: 40 vagy 80
OSZLOP.SZAM: 0 – 39 vagy 0 – 79 közötti érték
MELYIK: értéke 0 a karakter, és 1 az attribútumbájt esetében.

Képpontok leképezése grafikus módokban

Grafikus módokban a képpontokat bitsorozatonként tárolják, és a tárolt bitsorozat, valamint a képpontok között pontos megfeleltetés van. Általában a három séma valamelyikét használják a képernyő leképezésére grafikus módokban.

Az eredeti színes-grafikus adapternél a képernyő 200 soros, ezeket 0-tól 199-ig számozzuk. Az egyes sorokban a pontok száma a megjelenítési mód függvénye. A 4-es, 5-ös és 9-es mód közepes felbontású, soronként 320 ponttal. A 6-os és 10-es mód nagy felbontású, soronként 640 ponttal. A PCjr-nél bevezetett 8-as mód a szabványos színes-grafikus adapterrel nem

használható. A 8-as mód durva felbontású, soronként 160 ponttal. A képpontok oszlopait a durva, a közepes és a finom felbontásnál 0-tól rendre 159-ig, 319-ig, ill. 639-ig számozzuk.

A sorokat a tár összefüggő részében található *sorbankok* tárolják. A 4-es, 5-ös, 6-os és 8-as módban 2 bank van. Az elsőben találjuk a páros számú sorokat: 0., 2., 4 – 198., a másodikban a páratlan számú sorokat: 1., 3., 5 – 199. A 9-es és 10-es módban 4 bank van, ezekben így helyezkednek el a sorok:

1. bank 0., 4., 8., 12 – 196.
2. bank 1., 5., 9., 13 – 197.
3. bank 2., 6., 10., 14 – 198.
4. bank 3., 7., 11., 15 – 199.

Ezek a sorbankok két tekintetben hasonlítanak a szövegmódok oldalaira: egy bankon belül a sorok hézagmentesen követik egymást a tárban. Az egyes bankok viszont egész K címhatáron kezdődnek, így az egyes bankok végén kihasználatlan bájtok maradnak. Ellentétben azonban a képernyőoldalakkal, a képernyő egyszerre aktívan használja az összes sorbankot. Minden bank 8K-s, így az egyes bankok relatív kezdőcíme 0, 8K, 16K és 24K.

Mint a 4.12. táblázatban látható, az egyes képpontokhoz tartozó tár mennyisége a módtól függ. A 6-os mód 1 bitet használ, ez két színből választhat; a 4-es, 5-ös és 10-es mód 2 bitet, ezek négy színből választhatnak; a 8-as és 9-es pedig 4 bitet, ezek 16 színből választhatnak.

A különlegesen kezelt 10-es mód kivételével az egyes képpontok bitjeit a tár egymás utáni bitjeiről veszik elő. A 6-os módban pl. – ennél 1 képponthoz 1 bit tartozik – a képernyőtár első bájtja a képernyő első 8 képpontját határozza meg. Az első (a legmagasabb helyiértékű) bit határozza meg az első képpontot (és így tovább). A 4-es módban – 2 bit képpontonként – az egyes bájtok 8 bitje 4 képpontot vezérel. (Mind a három bitformátum megtalálható a 4.13. táblázatban.)

4.12. táblázat
A grafikus módok formátuma és tárigénye

Mód	Oszlopok	Színek	Bitek	Bank	Tár, K
4	320	4	2	2	16
5	320	4	2	2	16
6	640	2	1	2	16
8	160	16	4	2	16
9	320	16	4	4	32
10	640	4	2	4	32

A 10-es módban a bitek leképezése ettől eltérő. (L. a 4.14. táblázatot.) A 4-es és 5-ös módhoz hasonlóan a 10-es módnál is 2 bit tartozik egy képponthoz, de ez a 2 bit – a 4-es és az 5-ös módtól eltérően – nem egy bájt 2 szomszédos bitjén található. Ehelyett a 2 bit 2 szomszédos bájt megfelelő bitjében van. Az első bájtból származó bit a magasabb helyiértékű. Ha ezt a második bájt megfelelő bitjével kombináljuk, egy 0 és 3 közé eső színszámot kapunk.

A 4-es, az 5-ös, a 6-os és a 8-as módban egy sor képpontjai 80 bájtot foglalnak le; a 9-es és a 10-es módban pedig egy sor 160 bájtot foglal el.

4.13. táblázat
Az első képpontok bittérképe három grafikus megjelenítési módnál

Bit									Bit								
7	6	5	4	3	2	1	0	Képpont	7	6	5	4	3	2	1	0	Képpont
6-os mód									4-es és 5-ös mód								
X	.	.	.	.	.	.	.	1	X	X	.	.	.	.	.	.	1
.	X	.	.	.	.	.	.	2	.	.	X	X	.	.	.	.	2
.	.	X	.	.	.	.	.	3	.	.	.	.	X	X	.	.	3
.	.	.	X	.	.	.	.	4	.	.	.	.	.	.	X	X	4
.	.	.	.	X	.	.	.	5									
.	.	.	.	.	X	.	.	6	8-as, 9-es, 13-as és 14-es mód								
.	.	.	.	.	.	X	.	7	X	X	X	X	.	.	.	.	1
.	.	.	.	.	.	.	X	8	.	.	.	.	X	X	X	X	2

4.14. táblázat
Az első képpontok bittérképe a 10-es megjelenítési mód grafikus formájában

1. bájt								2. bájt								Képpont
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
X	.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	1
.	X	.	.	.	.	.	.	.	X	.	.	.	.	.	.	2
.	.	X	.	.	.	.	.	.	.	X	.	.	.	.	.	3
.	.	.	X	.	.	.	.	.	.	.	X	.	.	.	.	4
.	.	.	.	X	.	.	.	.	.	.	.	X	.	.	.	5
.	.	.	.	.	X	.	.	.	.	.	.	.	X	.	.	6
.	.	.	.	.	.	X	.	.	.	.	.	.	.	X	.	7
.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	X	8

A képernyő vezérlése

Általában a képernyő négyféleképpen vezérelhető, hasonlóan a többi számítógépes művelethez:

- A programnyelvek szolgáltatásainak igénybevételével (pl. a BASIC **SCREEN**-utasításával).
- DOS szolgáltatásokkal (☛ L. a 16. és a 17. fejezetet).
- ROM-BIOS videoszolgáltatásokkal (☛ L. a 9. fejezetet).
- A hardver közvetlen manipulációjával, a tár és a portok segítségével.

A programozási nyelvekben, a DOS-ban és a ROM-BIOS-ban rendelkezésre álló képernyős szolgáltatások a képernyő adatait automatikusan a képernyőtárban helyezik el. A különböző típusú szolgáltatások különböző szintű vezérlést tesznek lehetővé. A 16 ROM-BIOS-szolgáltatás különösen hatékony: szinte mindent biztosítanak, ami a kép előállításához, a kurzor irányításához és a képernyő tartalmának módosításához szükséges. (☛ Mind a 16 szolgáltatás részletesen megtalálható a 9. fejezetben.) Ha teljes egészében át akarjuk venni a képernyő fölötti ellenőrzést, akkor a szoftverszolgáltatások megkerülésével közvetlenül a képernyőtárba írhatjuk az adatokat. Ezzel csak akkor éljünk, ha nyomós okunk van rá.

Mielőtt a közvetlen műveletek mellett döntenénk, azt is vegyük figyelembe, hogy ez zavarja az ablakot használó rendszereket és a fejlettebb többtaskos operációs rendszereket. Mindent egybevetve, sok PC-s program közvetlenül állítja elő a képernyő képét – valójában olyan sok, hogy ez szinte szabvánnyá a kép előállításának bevett módjává vált. Bár a távoli jövőt is szem előtt tartva, nem a legbölcsebb a kép közvetlen előállítása – egyelőre mégis mindenki ezt teszi.

Alapjában véve azok a programok, amelyek közvetlenül beleírnak a képernyőtárba, nem keverhetők az ablakot használó rendszerekkel, mert a két program ugyanannak a tárnak a birtoklásáért küzdene, és összekevernék egymás adatait. Mivel azonban oly sok program használja a közvetlen képelőállítás módszerét, az IBM saját, többtaskos ablakot használó rendszere – a Topview – végtelen erőfeszítéseket tesz, hogy ilyen programokkal is együtt tudjon működni. Egy Topview-hoz hasonló rendszer egyszerűen úgy teljesíti ezt a feladatot, hogy külön példányban megőrzi a program képernyőtárát. Amikor a program fut, ez a példány a képernyő pufferébe kerül, amikor a program leáll, a képernyő pufferről új másolat készül. Ezzel a technikával a Topview eléri, hogy olyan programokat is futtasson, amelyek használják a képernyőtárat. Ezért azonban hatalmas árat kell fizetnie. A számítási önköltség és a tár önköltsége megnő, a program nem futhat a háttérben más programok mellett, és a képernyő tartalma nem kerülhet ablakba (azaz nem mozgatható el és mérete sem változtatható).

Néhány szó a kurzorról

A villogó kurzor a szöveges módokban az aktív képernyő-pozíciót jelzi. A kurzor valójában a pásztázó soroknak egy karakterpozíció szélességű csoportja. A karakterpozíció szélessége függ a képernyő adaptertől; a monokrom adapter 9 képpont széles, 14 sugársoros formátumot használ, a színes-grafikus adapter pedig 8 képpont széles és 8 sugársoros formát. (A monokrom adapter többlet sorai részletesebb rajzolatú karaktert eredményeznek. ☛ Erről meggyőződhetünk a *C* függelékben.)

Alaphelyzetben a kurzor az összes sugársort kihasználja, de az adott szűk tartományon belül tetszőlegesen megváltoztatható. Beállítható pl., hogy csak bizonyos sorokat rajzoljon ki, vagy egy lejjebb levő sor kirajzolása után egy feljebb levő sorban folytatódjon. Ez lehetővé teszi, hogy a karakter dobozában akárhol kialakítsunk egy összefüggő kurzort, vagy 2 részből álló kurzort hozzunk létre a karakter dobozának alján és tetején.

(☛ A 9. fejezetben részletes magyarázat található az ún. karakterdobozról, ill. a sugársorok és a karakter kapcsolatáról.)

Mivel a szöveges módokban használt villogó kurzort a hardver hozza létre, a szoftver befolyása elég korlátozott. Több mód is kínálkozik arra, hogy megváltoztassuk a formáját és a képernyőn elfoglalt helyét. A kurzor helyzetének megállapítására és beállítására felhasználhatunk bizonyos ROM-BIOS-szolgáltatásokat (☛ l. a 11. fejezetet), vagy direkt módon kiolvashatjuk, ill. beleírhatjuk a tárba. (☛ L. a hexa 450-es cím magyarázatát a 3. fejezetben.) A kurzor formáját a ROM-BIOS-szolgáltatásokkal hasonlóképpen kiolvashatjuk és beállíthatjuk. A formáját a tár közvetlen vizsgálatával is megállapíthatjuk. (☛ L. a hexa 460-as címhez fűzött magyarázatot a 3. fejezetben.) A legtöbb programozási nyelv is nyújt ilyen szolgáltatást.

Ha valaha meg akarjuk kerülni a hardver által vezérelt villogó kurzort (és sokan akarják), akkor a valódi kurzor megtalálása után használhatjuk az inverz attribútumot, a hexa 70-et. Ez egy tömör, nem villogó kurzort eredményez, amit más módon a CHR$(219)-es vagy a CHR$(254)-es tömör ASCII-karakterrel érhetünk el.

Eddig a szöveges módú kurzorral foglalkoztunk. A grafikus módokban a kurzor nem látszik, de beszélhetünk a kurzor logikai helyzetéről: ez a képernyő aktív pontját mondja meg. Akárcsak a szöveges módokban, itt is a ROM-BIOS szolgáltatásaival (vagy közvetlenül a hexa 450-es címről) tudhatjuk meg a kurzor helyzetét.

A kurzort grafikus módban úgy teszik láthatóvá – pl. a BASIC –, hogy más háttérszínt használnak egy karakternyi helyen (a kurzor helyén), vagy tömör ASCII karaktert írnak ki.

A programozó itt választás elé kerül: a képernyő közvetlen kezelése sebességi és teljesítménybeli előnyt nyújt; a BIOS vagy a magasabb szintű szolgáltatások igénybevétele pedig nagyobb rugalmasságot biztosít az ablakot használó rendszerekhez és az új képernyőadapterekhez való alkalmazkodásban. A szerző programjaiban mindkét technikát használja, és az igények szerint választja egyiket vagy másikat.

A hardver közvetlen vezérlése

A fejezetben eddig közölt ismeretek – főleg a képernyő-memória belső leképezése – azt a célt szolgálták, hogy segítségükkel könnyebben lehessen a videoinformációt közvetlenül a képernyőtárba írni. Emlékeztetünk azonban rá, hogy mindenfajta közvetlen programozás veszélyeket rejt magában. Rá fogunk jönni, hogy biztonságosabb és könnyebb, ha a lehető legmagasabb szintű eszközökkel vezéreljük a képernyőt. Alacsonyabb szintű eszközök (különösen a közvetlen belenyúlások) alkalmazása nagy veszéllyel járhat. Csak igen ritkán mondható el, hogy a közvetlen vezérlés biztonságos és megbízható.

A monokrom adapter I/O portjai

A monokrom adapter 4 I/O portot használ: a képernyő kontroll- és státuszportját, valamint a 6845-ös képernyőt vezérlő regisztereket.

A képernyő kontrollportja (hexa 3B8). A port 8 bitjéből hármat állíthatunk be: a nagyfelbontást, a videót és a villogást jelölő bitet. A monokrom adapternél a nagyfelbontás bitjének értéke mindig 1. A video- és a villogásbit kikapcsolja, ill. bekapcsolja a megjelenítést és a karakter villogását. A hexa 29-es érték kiküldése a megszokott értéket állítja be ezeken a biteken. (L. a 4.15. táblázatban.)

4.15. táblázat

A CRT-t vezérlő port kódolása

Bit 7 6 5 4 3 2 1 0	Jelentése
. X	Nagy felbontású mód: kötelezően 1
. X X .	Nincs felhasználva
. . . . X . . .	A 0-ás videojel letiltva, az 1 videojel engedélyezve
. . . X	Nincs felhasználva
. . X	A 0 villogtatás kikapcsolva, az 1 villogtatás bekapcsolva
X X	Nincs felhasználva

A képernyő státuszportja (hexa 3BA). A port 0-s bitje a vízszintes szinkronjel állapotát, 3-as bitje képernyőre tartó bitsorozatot mutatja. Bár mindkét bit kiolvasható, egyik sem mondható különösebben hasznosnak. A többi bit nincs felhasználva.

A 6845-ös képernyővezérlő (hexa 3B4 és 3B5). A 6845-nek 18 belső programozható regisztere van. Ezek szabályozzák pl. a függőleges és a vízszintes szinkronizáló jelek időzítését, a képernyő sorainak, ill. az egy sorban levő karaktereknek a számát! Mindössze 4 regiszter használata tekinthető biztonságosnak: a hexa 0A, 0B, 0E és 0F-é. A 0A és 0B regiszter határozza meg a kurzor kezdő és befejező sorát, a 0E és a 0F regiszter pedig a kurzor helyzetét a képernyőn. Az utóbbi érték 0 és 1999 közé esik. Mindkét funkció elérhető a ROM-BIOS szolgáltatásai között a 16-os megszakításon keresztül. Nem érdemes a többi értékkel próbálkozni: súlyos következményei lehetnek. (A monokrom képernyőt pl. tönkre lehet tenni a 6845-ös vezérlő helytelen programozásával.) Aki többet is meg akar tudni, az forduljon az IBM PC Technical Reference kézikönyvhöz.

A színes-grafikus adapter I/O portjai

A grafikus funkciók ellátásához a színes-grafikus adapternek a monokrom adapternél több I/O portja van. Felsoroljuk mind a 7 port fontosabb jellemzőit.

A módkiválasztó regiszter (hexa 3D8). Ennek a bájtnak a segítségével válthatunk át egyik megjelenítési módról a másikra. (L. a 4.16. táblázatot.)

A színkiválasztó regiszter (hexa 3D9). Ezzel a bájttal változtathatjuk a keret színét a szöveges módokban, ill. az előtér és a háttér színét a grafikus módokban. (L. a 4.17. táblázatot.)

A státuszregiszter (hexa 3DA). Ebben a regiszterben azok találnak hasznos adatokat, akik szeretik, ha a kép felfrissítése villogás- és havazásmentes. Amikor a 0-s bit értéke 1, akkor a kép megzavarása nélkül belenyúlhatunk a puffer tárba. Amikor a 3-as bit – a függőleges szinkron – értéke 1, akkor az elektronsugár függőlegesen visszafut, ilyenkor szabad a képet módosítani. A regiszterben található még a fényceruza két státuszjele. (L. a 4.18. táblázatban.)

A fényceruza reteszelő portjai (hexa 3DB és 3DC). Ha a két port valamelyikére írunk, az átbillent egy, a 6845-ös fényceruza bemenetéhez kapcsolódó, kétállású kapcsolót.

A 6845-ös képernyővezérlő (hexa 3D0 és 3D1). A vezérlő ugyanúgy működik a színes-grafikus adapterben, mint a monokrom adapterben.

4.16. táblázat
A módkiválasztó regiszter kódolása

Bit 7 6 5 4 3 2 1 0	Jelentése
. X	0 : 40 × 25 karakteres szöveges mód; 1 : 80 × 25 karakteres szöveges mód
. X .	0: szöveges mód; 1 : 320 × 200 pontos grafikus mód
. X . .	0: színes mód; 1: fekete-fehér mód
. . . . X . . .	0: videojel letiltva, 1: videojel engedélyezve
. . . X	1: 640 × 200 pontos fekete-fehér grafika
. . X	0: villogtatás kikapcsolva; 1: villogtatás bekapcsolva
X X	Nincs felhasználva

4.17. táblázat
A színkiválasztó regiszter kódolása

Bit 7 6 5 4 3 2 1 0	Jelentése
. X	Kék előtér, háttér vagy keret kiválasztása
. X .	Zöld előtér, háttér vagy keret kiválasztása
. X . .	Vörös előtér, háttér vagy keret kiválasztása
. . . . X . . .	Intenzitásérték kiválasztása
. . . X	Alternatív és intenzív paletta kiválasztása
. . X	0: 0-s paletta; 1: 1-es paletta
X X	Nincs felhasználva

4.18. táblázat
A státuszregiszter kódolása

Bit 7 6 5 4 3 2 1 0	Jelentése
. X	1: a tárhoz lehet fordulni a kép megzavarása nélkül
. X .	1: a fényceruza gombja le van nyomva
. X . .	0: fényceruza bekapcsolva, 1: fényceruza kikapcsolva
. . . . X . . .	1: függőleges képvisszafutás
X X X X	Nincs felhasználva

A képernyő vezérlése bonyolult feladat, és a PC megjelenítő funkcióinak állandó bővülése egyre tovább bonyolítja. Bárhogy is döntünk, ajánlatos először kísérletezéssel meggyőződni, vajon helyesen értjük-e a képernyő vezérlésének valamely részletét, mielőtt a programjainkba beépítenénk valamit.

Kompatibilitási szempontok

Több dolgot is szem előtt kell tartani, ha azt szeretnénk, hogy programjaink minden IBM személyi számítógéppel kompatíbilisek legyenek. A szabványos PC nem tud grafikus képet előállítani, ha csak a monokrom adapterrel rendelkezik. Ezen kívül a 8 – 10-es grafikus módok nem használhatók sem a színes grafikus adapter alaptípusánál, sem bővített változatainál – ezek csupán a PCjr színes rendszerében léteznek. Hasonlóképpen a 13 – 16-os módok az EGA-hoz tartoznak. Az előző megszorítások a színpaletták újra definiálására is érvényesek, lévén ezek is a PCjr és az EGA sajátosságai. Emlékezzünk arra is, hogy a PC- és XT-modellek monokrom adaptere különlegesen kezeli a szöveges módok színattribútumait.

Helyes, ha egy program a színek használatában, valamint a szöveges, ill. grafikus módok közötti választásnál vagy az IBM monokrom monitorhoz, vagy egy kompozit monokrom monitorhoz alkalmazkodik, bár ez utóbbinál a színek általában nem túl jól láthatók. Tartsuk szem előtt, hogy a PC-t gyakran használják kompozit monokrom monitorral, különösen ott, ahol elsősorban szöveggel dolgoznak (szövegfeldolgozás és könyvelés). Sok PC működik ilyen kiépítéssel, és bölcs dolog, ha programunk tekintettel van rájuk.

A fenti rendszereket úgy lehet figyelembe venni, hogy megállapítjuk a használt megjelenítési módot, és annak megfelelően járunk el. Olyan programok esetében, amelyek már úgyis assembly nyelvű szubrutinokon keresztül kapcsolódnak a BIOS-hoz, ezt legcélszerűbben a BIOS 15-ös szolgáltatásával érhetjük el. (☛ L. a 9. fejezetet.) Más programok számára a 15-ös szolgáltatás akadályt jelent. A probléma megkerülhető a tár 0000 : 0049 című rekeszének kiolvasásával – ez tárolja a megjelenítési módot (☛ L. a 3. fejezetben). BASIC-ben így olvashatjuk ki ezt a címet:

```
DEF SEG=0
VIDEO.MODE=PEEK(&H449)
```

A 7-es megjelenítési mód az IBM monokrom monitor használatát jelzi. A kompozit monokrom monitor használatát nem lehet automatikusan érzékelni; abban az esetben azonban, ha egy értelmes felhasználó a DOS **MODE** parancsát használja fel a színek elnyomására, akkor ezt a 0-s és 2-es módban programunk kiderítheti.

Több kompatibilitási követelményt is felállíthatunk, ha programjaink használhatóságát akarjuk áttekinteni a különböző IBM személyi számítógépek és a különböző típusú képernyők tekintetében. Ezek a követelmények nem teljesen ellentmondásmentesek, ami csupán azt tükrözi, hogy milyen belső ellentmondások vannak az IBM személyi számítógépek tervezése és a használható megjelenítési módok sokasága között. Ennek ellenére, most körvonalazunk néhány létező kompatibilitási alapelvet.

1. A csak szöveget használó képernyőkép növeli a kompatibilitást. Sok monokrom adapterrel felszerelt PC van, ami nem képes grafikus eredményközlésre. Amikor a tervezés során a „szöveg vagy grafika" kérdést mérlegeljük, két tényezőt kell figyelembe venni; egyik a grafika ellen, a másik mellette szól. Egyrészt – mint azt igen sok program igazolta – hatásos grafika hozható létre pusztán az IBM grafikus karaktereinek segítségével is. (☛ A *C* függelékben további adalékok találhatók a grafikus karakterek segítségével kialakított rajzokról.) Másrészt egyre általánosabb, hogy a számítógép grafikai képességekkel rendelkezik. A PCjr, az IBM Portable PC és a Compaq egyaránt beépített grafikával készül, ezért a jövőben várhatóan csökkenni fog a csak szöveges eredményt kiíró programok jelentősége, és a grafikát eleve beletervezhetjük programjainkba anélkül, hogy kompatibilitási gondoktól kellene tartani.

2. Minél kevésbé támaszkodik programunk a színekre, annál több számítógépen futtatható. Ez nem azt jelenti, hogy a kompatibilitás kedvéért el kellene kerülni a színek használatát;

hanem azt, hogy a kompatibilitás fokozása érdekében a színeknek csak hatásnövelő szerepük legyen, de tudjuk nélkülözni őket. Ha a program színek nélkül is boldogul, akkor monokrom képernyős gépeken is használható: pl. monokrom adapteres PC-ken, valamint a Compaqon és az IBM Portable PC-n. Ez utóbbiaknak beépített monokrom képernyője van.

3. Ha lehetséges, akkor szöveges módban a program működjön, mind 40, mind 80 karakter/sor módban is. Ez lehetővé teszi, hogy tv-vel is használhatók legyenek. A 40 oszlopos szélesség növeli az olvashatóságot, de csökkenti a kiírható információ mennyiségét. Ha nem kell sok információt kiírni, akkor esetleg eleve a 40 oszlopos formát választhatjuk; de általában legjobb alkalmazkodni a beállított szélességhez, vagy a felhasználótól megkérdezni: melyik formát használjuk. Ne felejtsük el, hogy a 40 oszlopos forma nem használható monokrom adapterrel felszerelt PC-n.

Ha ezeket az alapelveket a saját programunk szempontjából vizsgáljuk, akkor mérlegelni kell a széles körű kompatíbilitást, ill. a kevés képernyőtípus figyelembevételével elérhető egyszerűséget és kényelmet. A programozók nagyon gyakran hibát követnek el azzal, hogy a képernyők szűkebb választéka mellett döntenek, ezzel lecsökkentik azon számítógépek körét, amelyen programjuk használható.

5 Lemezes alapismeretek

Bevezetés • 88

A lemez fizikai felépítése • 88

A DOS lemezformátumai • 89

Szabványos DOS-formátumok • 89
Négyszeres sűrűségű formátumok • 90
A merevlemez formátuma • 91

A lemez logikai szerkezete • 92

A lemez szervezése a DOS alatt • 93
A hajlékonylemez helykiosztása • 93
A merevlemez helykiosztása • 95

A logikai szerkezet részletei • 96

A betöltőrekord • 97
A katalógus • 98
Az adatok területe • 103
A fájl allokációs táblázata • 104

Megjegyzések • 107

Bevezetés

A legtöbb számítógépes rendszeren mód van az információ tartós tárolására kazettás mágnesszalagon, hajlékonylemezen vagy merevlemezen. Ezeknek a tárolóknak sokféle lehet a mérete és a kapacitása, de működési elvük azonos: felületükön a mágnesesség segítségével kódolják az információt, olyan minták szerint, amit részben maga a berendezés, részben az azt irányító szoftver szab meg.

A PC-család 1981-es megjelenésekor a fő háttértár a szabványos 5 és 1/4 hüvelykes hajlékonylemez volt. Ez dupla sűrűségű, egyoldalas, „szoftszektoros" volt, és 160 kbájt információt tárolt. Az azóta eltelt időben az IBM megnövelte a hajlékonylemez tárolókapacitását, és bizonyos rendszereit ellátta 10 és 20 Mbájt-os merevlemezekkel is. Számíthatunk rá, hogy a jövőben az IBM (és mások is) továbbfejlesztik a lemezek technológiáját: ez többek között nagyobb kapacitású merevlemezeket és 3 és 1/2 hüvelykes mini hajlékonylemezeket jelent.

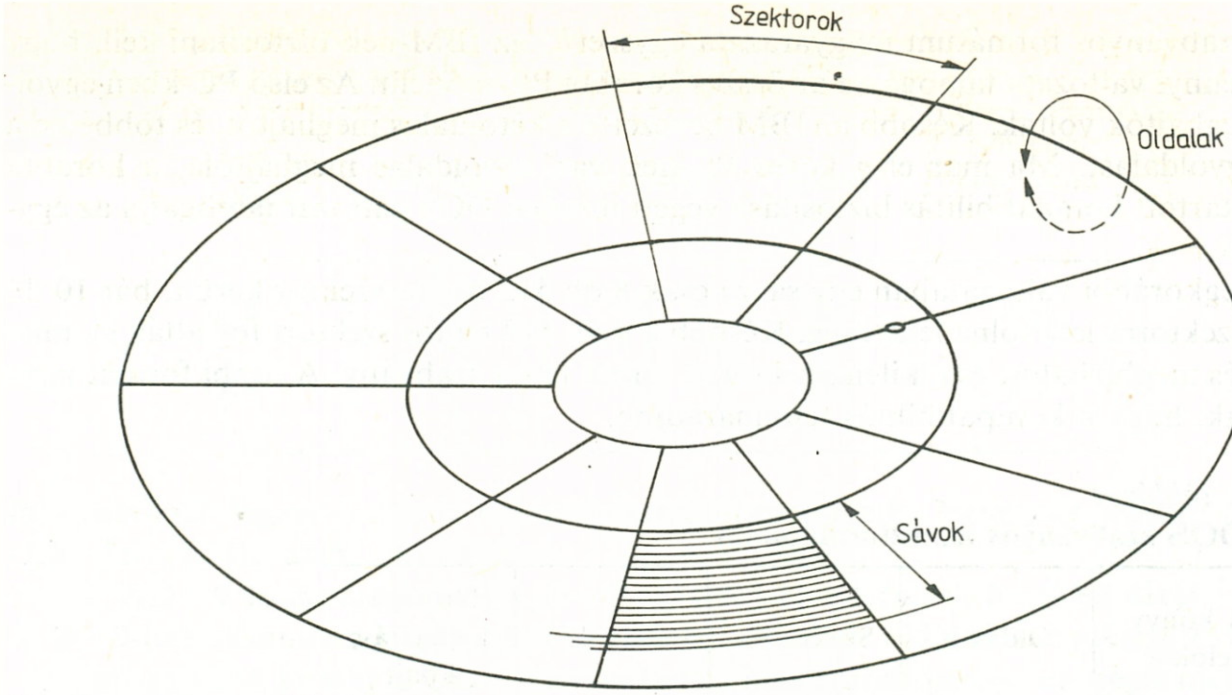
Bár a tároló berendezés típusa is fontos, a programozókat elsősorban az érdekli, hogyan is helyezkedik el a tárolt információ, és hogyan lehet kezelni. Ebben a fejezetben arra fogunk összpontosítani, hogyan szerveződik és tárolódik az információ a hajlékonylemezen, mivel ez a PC-család legáltalánosabban használt tárolóeszköze. Bár elsősorban a hajlékonylemezről fogunk beszélni, valójában bemutatjuk általában a lemez típusú tárolóberendezést. A fejezetben leírtak egyaránt érvényesek a RAM-lemezre (ez a lemez szimulálása a központi tárban), a hagyományos hajlékonylemezre, a merevlemezre, a minilemezre és a kazettás (kartridzs) lemezre.

A lemez fizikai felépítése

A lemezmeghajtó és az operációs rendszer meghatározzák a lemez tárolóképességét, a lemez felépítése (struktúrája) azonban alapvetően független ettől. Az adatokat a lemez felszínén mindig koncentrikus körök – sávok – sorozata mentén rögzítik. A sávokat további részekre – szektorokra – osztják. (L. az 5.1. ábrát.) A lemez egy-egy oldalán tárolható adatok mennyisége a sávok számától – az ún. sávsűrűségtől – és a szektor méretétől függ. A lemez sűrűségét lényegében a lemez típusa határozza meg: a szabványos dupla sűrűségű lemez 40 sávnyi adatot rögzít, az újabb négyszeres sűrűségű lemezek pedig 80 sávnyit.

A PC szabványos 5 és 1/4 hüvelykes hajlékonylemezeinél az egyes sávok helyzetét és a használható oldalak számát a hardver határozza meg, ily módon ezek rögzítettek és megváltoztathatatlanok. Egy sávon belül a szektorok helyét, méretét és számát a szoftver szabja meg. Ezért nevezik a PC hajlékonylemezeit „szoft-szektorosnak". A lemez szektorai a formattálás során kapják jellemző adataikat: méretüket és egy-egy sávon belüli szektorszámukat. A formattálást elvégezheti az operációs rendszer vagy a ROM-BIOS megfelelő szolgáltatása. A fejezet legnagyobb részében a DOS-formátumokat ismertetjük. Megtehetjük azonban azt is, hogy teljesen szokatlan formátumot hozunk létre a ROM-BIOS lemezes szolgáltatásainak segítségével, ezzel ugyanis hatékonyan megvédhetjük programunkat a másolás ellen. (☛ Az 5-ös szolgáltatást l. a 10. fejezetben.)

A szabványos PC BIOS által támogatott 5 és 1/4 hüvelykes hajlékonylemez szektorai 128, 256, 512 vagy 1024 bájtosak lehetnek. A DOS az 1-es változattól a 3.1-esig következetesen az 512 bájtos formát használta, és több mint valószínű, hogy ez a továbbiakban is így marad. Ennek ellenére minden olyan programnak, amely az 512 bájtos DOS-szektortól függ, vagy azt kihasználja, fel kell készülnie, hogy a szektor mérete a jövőben meg fog változni, valószínűleg megnő.



5.1. ábra. A lemez fizikai szerkezete

A hajlékonylemeznek természetesen két oldala van. A szektorokat és a sávokat tetszés szerint vagy csak az egyikre vagy mindkét oldalra tehetjük. A merevlemezes rendszerekben lehet egynél több tárcsa is, ezeknél tehát kettőnél több oldal is lehet. [XT] Az XT-modell 10 Mbájt-os lemezének pl. két tárcsája van, és mind a négy oldalt felhasználja.

A DOS lemezformátumai

A DOS-nak az IBM által használt korai változataiban csak kevés lemezformátumot lehetett használni, bár a meghajtóegységek maguk sok formátum írására és olvasására voltak képesek. A DOS 2.00-tól kezdődően minden újabb változatban a DOS-t csak néhány szabványos formattálási lehetőséggel látták el, ez azonban szinte tetszőleges fizikai formátum integrálására elegendő. (Látni fogjuk, hogy a logikai formátum jobban kötődik a DOS által definiált szabványhoz.) A rendszerbe integrálást az teszi lehetővé, hogy a DOS biztosítja a megfelelő eszközöket egy ún. installálható berendezésvezérlő megírásához. Ez nem más, mint egy olyan gépi kódú szubrutin, amelynek segítségével a lemezre különböző formátumokat lehet írni és olvasni, vagy összekötni a nem IBM-től származó lemezmeghajtót a rendszerünkkel. (☛ Az *A* függelékben találhatók további részletek az installálható berendezésvezérlőkről).

Mivel igen sok lemezformátum lehetséges, nincs rá mód, hogy mindet áttekintsük. Összesen hét, általánosan használt formátumot vizsgálunk meg: négy szabályos 5 és 1/4 hüvelykes hajlékonylemezest, egy speciális 5 és 1/4 hüvelykest, egy 3 és 1/2 hüvelykes mini hajlékonylemezest és egy merevlemezest. Ezek az általánosan használt formák elegendő támpontot nyújtanak ahhoz, hogy bármilyen típusú lemezzel boldoguljunk.

Szabványos DOS-formátumok

A négy leggyakoribb PC-s formátummal kezdjük. Az IBM ezeket használja az 5 és 1/4 hüvelykes lemezeknél. A négy formátum az oldalak számának és a sávonként használt szektorok számának különbségéből származik: van egy- és kétoldalas, ill. nyolc- és kilencszektoros. (☛ L. az 5.1. táblázatot.)

A négy szabványos formátum magyarázata egyszerű: az IBM-nek biztosítani kell, hogy a DOS valamennyi változata támogassa az összes korábbi PC-modellt. Az első PC-kben egyoldalas lemezmeghajtók voltak. Később az IBM bevezette a kétoldalas meghajtót, és többé nem is használt egyoldalast. Ma már elég kevés PC-nek van egyoldalas meghajtója, a korábbi gépekkel fenntartott kompatibilitás biztosítása végett minden DOS változat támogatja az egyoldalas formát.

A DOS legkorábbi változatában egy sávra csak 8 db 512 bájtos szektor került, bár 10 db ilyen méretű szektorra lett volna lehetőség. Később a 9 db 512 bájtos szektort fogadták el, mint biztonságost és megbízhatót, és a kilencszektoros forma lett a szabvány. A többi formát most is megtartották, hogy a kompatibilitás fennmaradjon.

5.1. táblázat
A DOS szabványos lemezformátumai

A könyv jelölése	Oldalak	Szektorok	Sávok	Névleges kapacitás, kbájt
S–8	1	8	40	160
D–8	2	8	40	320
S–9	1	9	40	180
D–9	2	9	40	360

A formátumok bővülése szorosan kötődik a DOS történetéhez és fejlődéséhez. A DOS 1.00-s változata csak az S–8-nak (S mint single, azaz egyoldalas) nevezett formát támogatta. A következő 1.10-es változat ezt kibővítette a D–8 (D mint double, azaz kétoldalas) formátummal. A 2.00-s változatnál vezették be a kilencszektoros formátumokat, az S–9-et és a D–9-et. A 2.10-es változatnál nem jelent meg új formátum, a DOS 3.00 azonban bevezette a hamarosan ismertetendő négyszeres sűrűségű formátumot.

Bár sok formátum van, csak kettő terjedt el igazán széles körben: az S–8 és a D–9. Az S–8 formát nevezhetjük a közös nevezőnek, ezért a kereskedelmileg forgalmazott programok ezt a formátumot használják. Ennek a formátumnak a használata biztosíték arra, hogy a lemezt a DOS minden változata el tudja olvasni. Ez a gyakorlat mégis lassan kimegy a divatból, különösen azoknál a cégeknél, ahol nagyméretű programokat forgalmaznak – ezeknek sok hely kell a lemezen. A D–9 a legnagyobb kapacitású formátum, amit a legtöbb 5 és 1/4 hüvelykes meghajtó használ, ezért a legtöbben ezt alkalmazzák munkalemezeiken. A D–8 és S–9 formátum használata nem ennyire általános, de alkalmanként előfordulnak.

Négyszeres sűrűségű formátumok

Megfigyelhető, hogy a négy szabványos formátumnál az egyetlen közös tényező a sávok száma: ez mindegyiknél 40. Ennek az az oka hogy a PC-családban leggyakrabban használt 5 és 1/4 hüvelykes meghajtókat 40 adatsáv írására és olvasására tervezték. Néhány 5 és 1/4 hüvelykes hajlékonylemezes meghajtó, ill. a legtöbb 3 és 1/2 hüvelykes meghajtó azonban ugyanazon a felületen 80 adatsávot tud rögzíteni. Ezt a meghajtótípust és a hozzá használt lemezt gyakran négyszeres sűrűségűnek nevezik. A lehetséges négyszeres sűrűségű formátumok közül csak a

5.2. táblázat

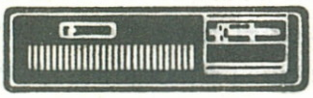
A négyszeres sűrűségű lemezformátumok

A könyv jelölése	Oldalak	Szektorok	Sávok	Névleges kapacitás, kbájt
QD – 9	2	9	80	720
QD – 15	2	15	80	1200

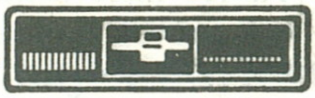
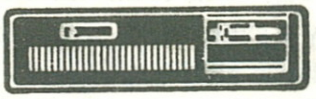
PC-családnál leggyakrabban használt kettőt ismertetjük. Nevezzük ezeket QD – 9-nek és QD – 15-nek. (L. az 5.2. táblázatot.)

A QD – 9 nagyon hasonlít a D – 9-re, azzal az eltéréssel, hogy 80 sávja van 40 helyett. A D – 9-hez hasonlóan a QD – 9 is kétoldalas, mindkét oldalon sávonként 9 szektorral.

Bár az IBM kerülte a QD – 9 használatát, más számítógépeknél mégis megtaláljuk, pl. a Data General DG – 1 összecsukható gépénél, a PC-család közeli rokonánál. A DG – 1 a szokásos 5 és 1/4 hüvelykes lemezek helyett 3 és 1/2 hüvelykest használ, de a formátum logikai szerkezete azonos. A minilemezek meghajtói négyszeres sűrűségűek, a lemezek mégsem csak QD – 9 formátumra formattálhatók, hanem S – 8, S – 9, D – 8 és D – 9 formátumra is. Négyszeres sűrűségű meghajtók nem szabványos berendezésként a PC-hez is csatolhatók a DOS-berendezésvezérlő segítségével. (📖 Az *A* függelékben olvashatók részletek a berendezésvezérlőkről.) Sokan úgy vélik, hogy ez a formátum széles körben el fog terjedni, tehát valódi érdeklődésre tarthat számot.

AT  Az AT-nál használt QD – 15 formátum beleilleszkedik az eddig ismertetett struktúrákba: oldalanként 80 sáv, szabványos 512 bájtos szektorok. A QD – 15-nek az a sajátossága, hogy minden sáv mindkét oldala 15 szektoros, a szokásos 8 vagy 9 helyett. Ilyen sok szektort csak azért lehet egy sávra elhelyezni, mert az AT különlegesen nagy kapacitású hajlékonylemezt használ. Ennek mágneses bevonata eltér a közönséges lemezétől. Csakis ezek a lemezek – amelyek külalakra nem különböznek a szabályos 5 és 1/4 hüvelykes lemeztől – és a különlegesen nagy kapacitású meghajtók képesek a QD – 15 formátum használatára.

A merevlemez formátuma

XT  AT  A nagy kapacitású merevlemezes rendszerek – pl. az XT 10 Mbájt-os vagy az AT 20 Mbájt-os lemez – néhány különleges problémát és lehetőséget vet fel.

Minden lemeznél két meghatározó szempont van: a fizikai formátum és a logikai formátum. A lemez fizikai formátuma meghatározza: hány bájtos egy szektor, hány szektor van sávonként (merevlemeznél a sávot cilindernek nevezik), ill. a sávok és a felületek számát. A logikai formátum azt jelenti, milyen az adatok szervezése a lemezen, és hova kerülnek a különböző jellegű információk.

Amikor a DOS-szal (vagy bármilyen más operációs rendszerrel) a hajlékonylemezt formázzuk, akkor a fizikai és a logikai formátumot is létrehozzuk, de nem vesszük észre, hogy a kettő között különbség van. A hajlékonylemezzel ellentétben a merevlemez fizikai formátuma

5.3. táblázat
Az XT és az AT merevlemezeinek fizikai formátuma

A könyv jelölése	Oldalak	Szektorok	Cilinderek	Névleges kapacitás, kbájt
XT	4	17	306	10
AT	4	17	615	20

már kész, amikor mi megkapjuk, azt már a gyár létrehozta. (L. az 5.3. táblázatot.) A gyárilag beállított fizikai formátum mellől a lemez logikai formátuma hiányzik. Ezt létre kell hozni, mielőtt az operációs rendszer használni tudná. Ez két lépésben történik. Először a merevlemezt *partíciókra* kell felosztani. Ezek tartalmazzák az általunk használt operációs rendszer adatait és programjait. (Egy merevlemezzel több operációs rendszert is használhatunk ☛ l. a merevlemez particionálását). Ezután az egyes particiók szervezését kell defíniálni, hogy az operációs rendszerek megtalálják adataikat a saját partíciójukon belül. Általában a lemeznek ezt a „megszervezését" nevezik formattálásnak.

A lemez logikai szerkezete

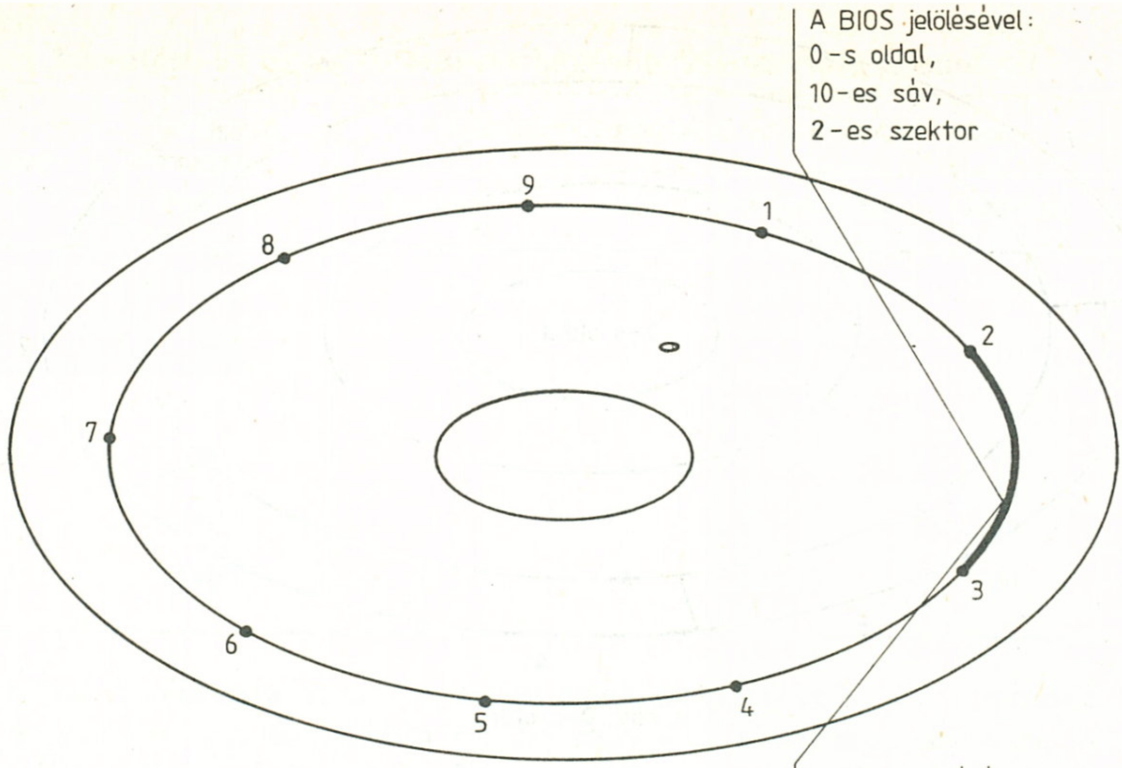
A lemez típusától függetlenül a DOS lemezeinek logikai formázása azonos: a lemez oldalát és szektorát egységes rendszer szerint megszámozzák, bizonyos szektorokat pedig fenntart a DOS a lemezes műveleteket vezérlő programok és indexek számára. Mielőtt rátérnénk arra, hogyan kezeli a DOS a lemez tárolóterületét, röviden ismertetni kell, milyen jelölést használ a DOS és a BIOS az információ helyének meghatározására.

Mint korábban már láttuk, az 5 és 1/4 hüvelykes lemeznek 40 sávja van. Ezt 0-tól (a legkülső sávtól) 39-ig, a középponthoz legközelebb eső, legbelső sávig számozzuk. Más formátumú lemezeknek esetleg több sávja van. A négyszeres sűrűségű lemez sávjait pl. 0-tól 79-ig számozzuk, [XT] az XT merevlemezén a cilindereket 0-tól 305-ig, [AT] az AT lemezén pedig 0-tól 614-ig.

A kétoldalas hajlékonylemez két oldalát 0-val és 1-gyel jelöljük. (A kétoldalas lemezmeghajtó két fejét is hasonlóan.) Az egyoldalas lemez egyetlen felülete a 0-s. [XT] Az XT merevlemeze négyoldalas – tehát négy feje van – ezeket 0-tól 3-ig számozzuk.

A hajlékonylemez szektorait 1-től 8-ig vagy 9-ig számozzuk. [XT] Az XT merevlemezén 1-től 7-ig tart a számozás. Figyeljük meg, hogy a szektorok számozása 1-től, a sávok és oldalak számozása viszont 0-tól kezdődik!

A szektorokat a BIOS egy háromdimenziós koordináta segítségével találja meg. Ennek komponensei a sáv száma (vagy cilinderszám), az oldal száma (vagy fejszám) és a szektor száma. A DOS viszont szektorszám alapján keresi meg az információt; a szektorokat kívülről befelé szekvenciálisan számozza. (L. az 5.2. ábrát.) A számozás a lemez 1. szektorával kezdődik: a 0. oldal 0. sávjának 1. szektorával. Ezt követik ennek az oldalnak és sávnak a szektorai. Kétoldalas lemeznél a 0. oldal 0. sávjának 9. szektorát, az 1. oldal 0. sávjának 1. szektora követi. A sorrend egy oldal egy sávjának szektorain halad végig, majd a következő oldalon ugyanazon a sávon – tehát előbb sorra vesszük egy sáv szektorait az összes oldalon, mielőtt a következő sávra térnénk át.



5.2. ábra. A szektorok jelölése a ROM-BIOS-ban és a DOS-ban

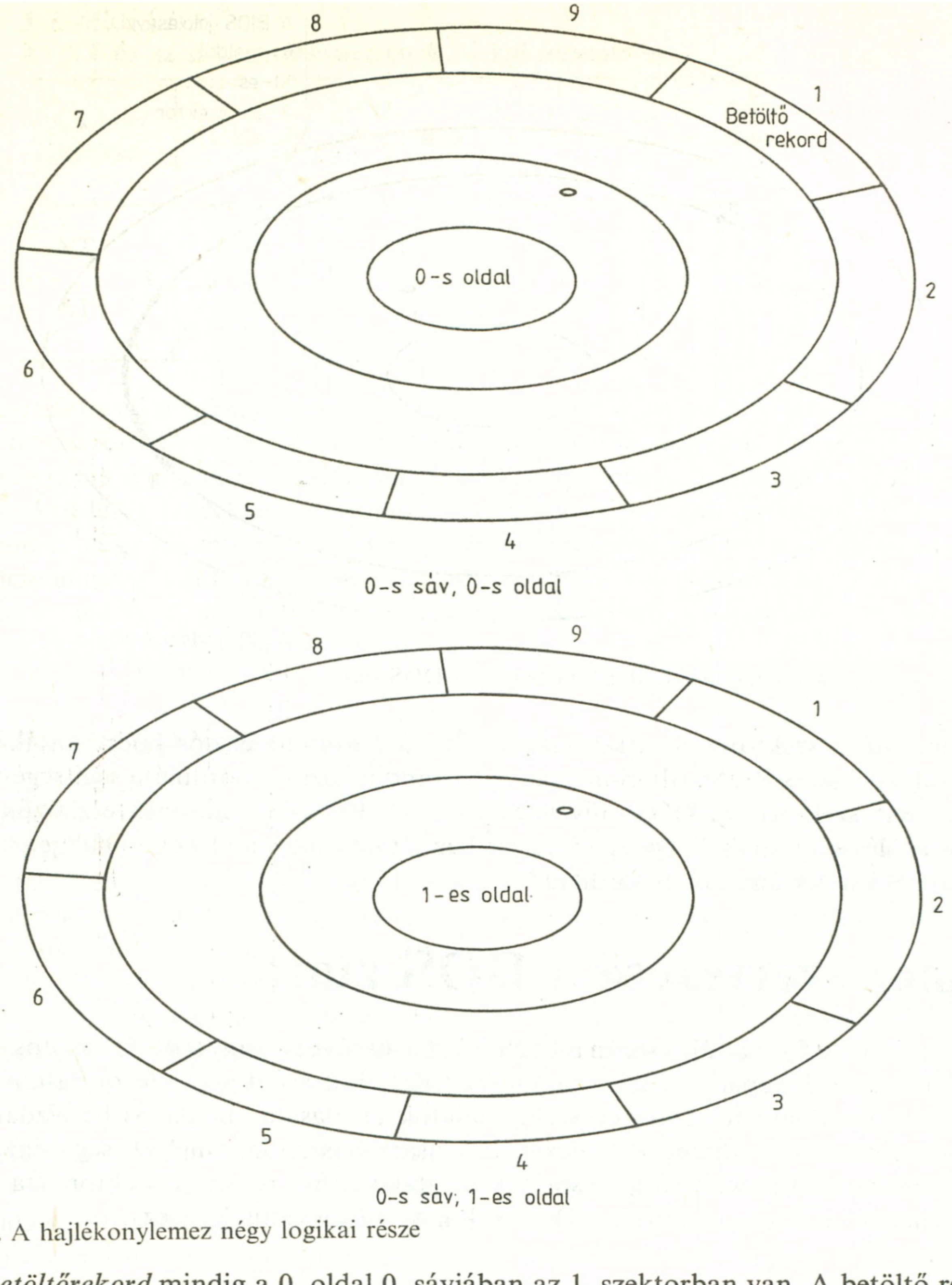
Egy megadott szektorra hivatkozhatunk akár a háromdimenziós koordinátákkal, akár sorszámával. Az összes ROM-BIOS-művelet a háromdimenziós koordináta segítségével találja meg a keresett szektort. A DOS-műveletek és a DEBUG-hoz hasonló eszközök a DOS sorszámos jelölését használják. (☛ A 15. fejezetben látható, hogyan lehet a DOS jelölésmódját a ROM-BIOS-éra átszámítani és fordítva.)

A lemez szervezése a DOS alatt

Amint már láttuk, a formattálás során a DOS mind a 40 sávot 8 vagy 9 db 512 bájtos szektorra osztja. A nyers tárolókapacitást tekintve ez 368 640 bájtot jelent a D – 9 formátumnál. Ez a terület nem használható föl teljes egészében adatok tárolására; bizonyos hányadát a DOS olyan rendszerszervezési adatok és indexek tárolására használja, amelyek segítségével egyes szektorok, ill. szektorok közötti kapcsolatok megtalálhatók. A sávok szektorokra osztásán kívül, a lemez formattálása közben, a DOS sok más műveletet is elvégez.

A hajlékonylemez helykiosztása

A formattálási eljárás során a lemez szektorai – a felhasználási módnak megfelelően – 4 részre lesznek felosztva. Az egyes részek a tárolás sorrendjében a következők: a betöltőrekord, az állomány kijelölési táblázat (FAT, file allocation table), a katalógus és az adatterület. (L. az 5.3. ábrát.) Az egyes részek mérete a formátum függvénye de szerkezetük és sorrendjük nem. A merevlemezek – pl. az XT 10 Mbájt-os lemeze – ugyanezt az alapvető sémát követik, bár a partíciókra osztható merevlemez bonyodalmakat okoz, mert a partíciók mérete közvetlen befolyással van az egyes részek méretére. (☛ A merevlemezek particionálását később ismertetjük.)



5.3. ábra. A hajlékonylemez négy logikai része

A betöltőrekord mindig a 0. oldal 0. sávjában az 1. szektorban van. A betöltő rekordban egyebek között egy olyan program található, amely elindítja az operációs rendszer betöltését egy hajlékonylemezről, ha azon rajta van az operációs rendszer. A betöltőrekord minden hajlékonylemezen megtalálható, akkor is, ha az operációs rendszer nincs rajta. Az indítóprogramtól eltekintve a betöltőrekord pontos tartalma a formátumtól függ.

Az állománykijelölés táblázata – más néven *FAT* – a betöltőrekordot követi. Rendszerint a 0. oldal 0. sávjának 2. szektorában kezdődik. A FAT tartalmazza a lemez formátumának hivatalos feljegyzését, és megadja az állományok által használt szektorok helyét. A DOS a FAT segítségével tartja számon az adatterület felhasználását. A táblázat minden egyes eleme egy kód segítségével jelzi, hogy melyik területet használják, melyik szabad, és melyik használhatatlan (lemezhiba miatt). A FAT tartja ellenőrzés alatt a lemez teljes felhasználható adatterületét, és

5.4. táblázat

A szektorok felosztása az egyes szabványos hajlékonylemezes formátumoknál

Formátum	Szektorok száma összesen	Adminisztrációs célra használt szektorok				Adat-szektorok
		Betöltő	FAT	Katalógus	Összesen	
S–8	320	1	2	4	7	323
D–8	640	1	2	7	10	630
S–9	360	1	4	4	9	351
D–9	720	1	4	7	12	708
QD–9	1440	1	10	7	18	1422
QD–15	2400	1	14	14	29	2371

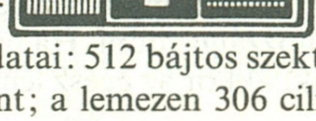
két példányban tárolja arra az esetre, ha az egyik megsérülne. A FAT mindkét példánya a szükséges számú szektort foglalhatja el: 2-t vagy 4-et a hajlékonylemezen, 14-et QD – 15 formájú hajlékonylemezen, legföljebb 16-ot az XT merevlemezén és legföljebb 82-t az AT merevlemezén. A különféle típusú merevlemezeken a FAT mérete függ a partíció méretétől.

A katalógus a lemez következő egysége. Ezt tekinthetjük a lemez tartalomjegyzékének. A lemez minden állományához (fájljához) egy azonosító bejegyzést tartalmaz, különböző lényeges információkkal, többek között a fájl nevével és méretével. A bejegyzés egyik része a fájlt tartalmazó első szektorcsoportra mutató szám – a FAT-ban is ez a szám szerepel, mint a fájl eleje. A katalógus mérete a lemez formátumától függ. Egyoldalas lemezen 4 szektort, kétoldalas lemezen 7 szektort foglal el. Merevlemeznél a katalógus mérete – a FAT-hoz hasonlóan – a partíció méretétől függ.

Az adatterület foglalja el a lemez legnagyobb részét. A katalógus végétől az utolsó szektorig terjed. Feladata az adatok tárolása, a másik három rész is ezt a funkciót támogatja. Az adatterület szektorait igény szerint osztják ki ún. klaszterekben. A klaszter mérete függ a formátumtól. Egyoldalas lemezen a klaszter hossza 1 szektor, kétoldalas lemezen 2 szomszédos szektor. Nagyobb kapacitású hajlékonylemezen esetleg több szektor alkot 1 klasztert. [AT] Az AT 20 Mbájt-os merevlemezén a klaszter 4 szektorból áll. [XT] Az XT 10 Mbájt-os lemezén 1 klaszter 8 szektoros.

A merevlemez helykiosztása

Merevlemezes rendszereknél a DOS partíció méretétől függ a FAT, a katalógus és az adatterület mérete. A betöltőrekord (a partíció méretétől függetlenül) 1 szektort foglal el, ezzel tehát a továbbiakban nem foglalkozunk.

XT 

A DOS helykiosztásának megismeréséhez most 3 különböző partícióméretet vizsgálunk meg. Példáink alapja XT 10 Mbájt-os merevlemeze. Adatai: 512 bájtos szektorok, 17 szektor cilinderenként és oldalanként, 4 oldal és fej cilinderenként; a lemezen 306 cilinder van.

Az 5.5. táblázatban konkrét adatok találhatók 3 különböző DOS partícióméretnél: azaz a 305 cilinderesnél, ekkor az egész lemez a DOS-é; a 100 cilinderesnél és az 5 cilinderesnél. Ezekből az adatokból interpoláció segítségével meghatározható a helykiosztás más partícióméretekhez is.

5.5. táblázat
Három partíció helykiosztása az XT 10 Mbyte-os merevlemezén

A területek megnevezése	A partició mérete (cilinder)		
	305 (összes)	100	5
A FAT mérete (szektor)	8	5	1
A katalógus mérete (szektor)	32	16	4
A katalógus bejegyzések száma	512	256	64
A klaszter mérete (szektor)	8	4	1
A klaszterek száma	2587	1699	333
Az adatterület mérete, K	10 348	3372	166,5

A logikai szerkezet részletei

Itt az ideje, hogy részleteiben is megismerjük a lemez négy részét: a betöltőrekordot, a katalógust, az adatterületet és a FAT-ot.

A merevlemez particionálása

Minden operációs rendszer a maga sajátos módján formázza és kezeli a lemez tárolóterületét. Ezek nem kompatíbilisek más operációs rendszereknél használt módszerekkel. Bár a PC-családban jelenleg a DOS a legelterjedtebb operációs rendszer, mégsem csak a DOS-t használják. Sőt, az is elképzelhető, hogy az elkövetkező években a DOS-t kiszorítja valamelyik új operációs rendszer, pl. a XENIX.

Előfordulhat, hogy több operációs rendszer használja ugyanazt a lemezt, ezért kidolgoztak egy megfelelő rendszert a lemez particionálására (felosztására), miközben mindegyik operációs rendszer megkapja a maga részét. Egy partíció szomszédos cilinderekből áll, méretét a felhasználó dönti el, és az operációs rendszer rendezi el. Még azelőtt partíciókra kell osztani a merevlemezt, mielőtt egy operációs rendszer használni kezdené. A particionálás végeztével az egyes partíciókat az egyes operációs rendszerek megfelelő eljárásaival formázni kell. Általában egy operációs rendszer egy partíciót használ. Előfordul azonban, hogy egyes kereskedők a merevlemezt két külön lemezmeghajtónak megfelelő partíciókra osztják, és mindkét partíciót a DOS használja. Szerencsére ez csupán kivétel.

A DOS FDISK programja hozza létre a partíciókat, és egyiket DOS-partíciónak jelöli ki. Megadhatjuk, hogy a DOS-partíció foglalja el a teljes lemezt, vagy annak csak egy részét. Az operációs rendszer tetszőleges méretű partícióhoz képes hozzáigazodni. Ha nem számoltunk azzal, hogy más operációs rendszert is használni fogunk, akkor az egész lemez lehet a DOS partíciója.

A partíciók számát és méretét akármikor meg lehet változtatni. Figyelembe kell azonban venni, hogy a partíció megváltoztatása megsemmisíti a partícióban tárolt adatokat. Ezért először az adatokat ki kell menteni, utána a lemezt újra lehet particionálni, majd az adatokat vissza kell tölteni.

A partíció megformázása

A merevlemez logikai struktúrája kétszintű. Az egyik szintet a lemez partíciókra osztása jelenti, ez minden operációs rendszerre nézve közös. A másik szint az egyes partíciókban tárolt információ sajátos formája és helye, ez minden operációs rendszernél más. Miután az FDISK-kel létrehoztuk a DOS partícióját, ezt a partíciót a DOS **FORMAT** parancsával kell megformázni ahhoz, hogy létrehozzuk a DOS működéséhez szükséges logikai szerkezetet. Hasonlóan ahhoz, amikor egy DOS-lemez 1. szektorában egy betöltőrekordot találunk, amely az indítóprogramot tartalmazza a lemez általános adataival együtt. Egy partíciókra osztható lemez első szektora egy ún. fő betöltőrekordot tartalmaz. Ebben található egy fő indítóprogram, valamint az, hogy milyen partíciókra van a lemez felosztva. Ez a rekord a következő adatokat tárolja a particionálásról: a partíciók számát (ez gyakran csak egy), az egyes partíciók méretét és helyét, a partíció típusát (melyik az aktív partíció). A fő indítóprogram megállapítja, hogy melyik az aktív partíció, majd átadja a vezérlést a megfelelő partíciót indító – vagy betöltő – programnak.

Az eddigi általános megjegyzések minden olyan lemezre érvényesek, amely alkalmas arra, hogy particionálás után több operációs rendszer használja. A particionálás mechanizmusát az XT 10 Mbájt-os lemezén fogjuk szemléltetni.

Az XT merevlemezének fő betöltőrekordjában egy olyan partíciós táblázat található, amelyben 4 partíciónak van hely. A táblázat minden elemében van egy jelzés arról, hogy melyik az aktív partíció, és az ún. *ID* (azonosító) bájt jelzi, hogy melyik operációs rendszerhez tartozik. A DOS jele: 01.

A partíció helyét és méretét a fő betöltőrekord táblázata két, egymással egyenértékű módon tárolja. Az első megadja a partíció kezdő és záró pozícióját – azaz a partíció első és utolsó szektorának cilinderét –, sáv-, fej-, oldal- vagy szektorszámát. A másik módszer a partíció első szektorának a lemez első szektorához viszonyított relatív sorszámát adja meg, ezt követi a partíció összes szektorainak száma.

Minden partícióhoz egymás utáni cilinderek tartoznak. A partíció a hozzá tartozó első cilinder első szektorától az utolsó cilinder utolsó szektoráig tart. Ettől kicsit eltér a partíció, amelyik a lemez legelső cilinderét használja. Ezen a cilinderen a partíció a 2. szektornál kezdődik, mivel az elsőt a lemez fő betöltőrekordja foglalja el.

A betöltö rekord

A betöltőrekord mindenekelőtt egy rövid gépi kódú programot tartalmaz: ez indítja el a DOS betöltését a tárba. Először a program ellenőrzi, vajon a lemez rendszerlemez-e, azaz tartalmazza-e az IBMBIO.COM és az IBMDOS.COM fájlokat. A program az eredménytől függően halad tovább.

A betöltőprogramot a DEBUG segítségével vizsgálhatjuk meg. Ez ugyanis alkalmas arra, hogy a lemez tetszőleges szektorát beolvassa, és arra is, hogy a gépi kódot szimbolikus kódra fordítsa vissza. Aki közelebbről meg akarja vizsgálni a betöltőprogramot, és nem retten vissza a DEBUG nyers parancsformáitól, írja be a következőket:

```
DEBUG
L 0 0 0 1    ; töltsd be az 1. szektort
U 0 L 2      ; fordítsd vissza és listázd az első 2 bájtot
U 2E         ; fordítsd vissza és listázd a bájtokat 2E-től.
```

Ezekkel a parancsokkal meg lehet vizsgálni az *A* lemezegység betöltőprogramjának első utasításait.

5.6. táblázat
A betöltőrekord paraméterei

Kezdő-cím	Hossz	Jelentése
3	8 bájt	Rendszerazonosító (pl. IBM 2.1)
11	1 szó	A bájtok száma szektoronként (pl. 512, hexa 0200)
13	1 bájt	A szektorok száma klaszterenként (pl. 01 vagy 02)
14	1 szó	A lemez elején a fenntartott szektorok száma hajlékonylemeznél: 1
16	1 bájt	A FAT példányainak száma hajlékonylemez-nél: 2
17	1 szó	A katalógus bejegyzések száma a gyökérben (pl. 64 vagy 12)
19	1 szó	A szektorok teljes száma a lemezen (pl. 720 D – 9 esetén)
21	1 bájt	Formátumazonosító (pl. FF, FE, FD vagy FC l. 5.11. táblázatot)
22	1 szó	A FAT által elfoglalt szektorok száma (pl. 1 vagy 2)
24	1 szó	A szektorok száma sávonként (pl. 8 vagy 9)
26	1 szó	Az oldalak (fejek) száma (pl. 1 vagy 2)
28	1 szó	A különleges (fenntartott) szektorok száma

🕮 MEGJEGYZÉS: A 2E-nél kezdődő kód csak a szabványos IBM PC betöltőprogramjának a kezdete. Az első szektor második bájtja minden modellnél megmondja a betöltőkód kezdetét. Ha tehát nem PC-t használunk, akkor a harmadiknak beírt parancsnál 2E helyett azt az értéket kell használni, amit a második parancs írt ki a JMP-utasításban.

Az S – 8 és D – 8 formátumok kivételével a 4. bájttól kezdve néhány alapvető fontosságú paramétert találunk a betöltő rekordban. (L. az 5.6. táblázatot.) Ezek a paraméterek részei annak a BIOS paraméterblokknak, amelynek segítségével a DOS felügyeli a lemez típusú berendezéseket. A betöltőprogram az első 3 bájtban (0., 1., 2. bájt) kezdődik és a BIOS-paraméterblokk mögött folytatódik. A DOS 2.00 változattól kezdve a betöltőrekordot a 2 bájtos hexa 55AA aláírás zárja.

A katalógus

A lemezkatalógusok (directoryk) feladata az, hogy a lemezen tárolt fájlokról a legfontosabb adatokat tárolják. Ilyenek: a fájl neve, mérete, az első FAT-elem, a létrehozás dátuma és ideje, valamint néhány speciális fájlattribútum. (L. az 5.7. táblázatot.) Csak egy olyan információ van, amely nem található meg a katalógusban: a fájlt alkotó klaszterek pontos helye. Ezeket a FAT-ban találjuk.

A lemez minden fájljához tartozik egy bejegyzés a katalógusban, itt a fájlok közé soroljuk az alkatalógusokat és a lemez azonosító címkéjét is. A katalógus elemei 32 bájt hosszúak, így

5.7. táblázat
A katalógusbejegyzés részei

Mező	Kezdő-cím	Tartalma	Mérete, bájt	Formátuma
1	0	Fájlnév	8	ASCII-karakterek
2	8	A fájlnév kiegészítése	3	ASCII-karakterek
3	11	Attribútum	1	Bitek, kódolva
4	12	Fenntartott	10	Nincs felhasználva: nullák
5	22	Idő	2	Szó, kódolva
6	24	Dátum	2	Szó, kódolva
7	26	Az első FAT-bejegyzés	2	Szó
8	28	Fájlméret	4	Egész

egy szektorba 16 elem fér el. Az egyoldalas lemez katalógusának 4 szektorában 64 elem fér el. A kétoldalas lemezek 7 szektorában 112 bejegyzés fér el. Az alkatalógusokat közönséges fájlként kezelik, és a számuk nincs korlátozva. (☛ Az alkatalógusokról még bővebben fogunk szólni.) A katalógusok 32 bájtos elemei 8 mezőre oszthatók.

Az 1. mező: a fájl neve

A bejegyzés első 8 bájtja a fájl nevét tartalmazza ASCII kódban. Ha a fájl neve 8 karakternél rövidebb, akkor a jobb széle szóközökkel (**CHR$(32)**) egészül ki. A betűk nagybetűk, mivel a kisbetűket nem ismeri fel helyesen. A fájl neve általában nem tartalmazhat közbeékelt szóközt, mint pl. AA BB. A legtöbb DOS-parancs (a **DEL** vagy a **COPY**) nem ismeri fel a fájl nevét, ha közbeékelt szóközt tartalmaz. A BASIC azonban boldogul az ilyen fájlnevekkel és a DOS szolgáltatásai is. (☛ L. a 15. és 16. fejezet.) Ezek a lehetőségek érdekes trükköket tesznek lehetővé, pl. nehezen letörölhető fájlokat lehet így létrehozni.

A fájlnévmező első bájtján bizonyos esetekben egy különleges kód áll. Az egyáltalán nem használt bejegyzések első bájtja: 00. Ennek segítségével a DOS meg tudja állapítani, hogy mikor ért a katalógusban a bejegyzések végére anélkül, hogy a teljes területet végig kellene vizsgálnia. Ez a konvenció a DOS 2.00 óta érvényes, a korábbi változatokra nem.

Ha a fájlnév első bájtjának értéke E5, ez általában azt jelenti, hogy a fájlt törölték. Mivel azonban a DOS 1-es változatai nem használták a 00 kódot a még sosem használt bejegyzés jelölésére, az ebben a bájtban található E5 azt is jelentheti, hogy ezt a bejegyzést még nem használták.

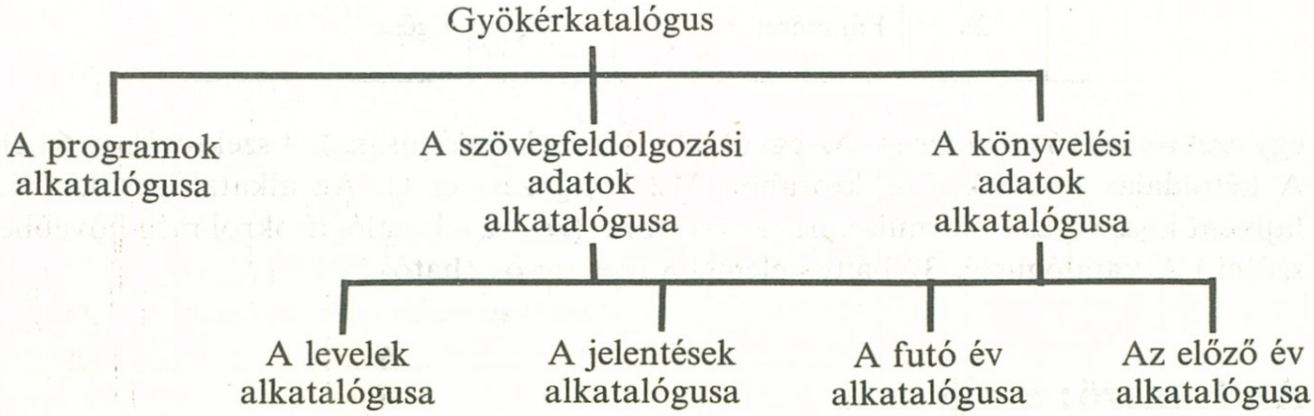
Mikor egy fájlt törlünk, két dolog történik: a fájl nevének első bájtjába E5 kerül, és a fájl számára fenntartott helyet jelző lánc törlődik a FAT-ból. (☛ Erről a FAT-ról szóló részben még beszélünk.) A fájllal kapcsolatos többi információ megmarad, beleértve a fájlnév többi részét, méretét, sőt a kezdő klaszter számát is. Az elveszett információ megfelelően kifinomult módszerekkel visszanyerhető – feltéve, hogy a törölt fájl bejegyzésének területét nem használta fel másik fájl.

Alkatalógusok

Kétfajta katalógus van: gyökér- és alkatalógus. Mindkét fajta tartalma és alkalmazása alapjában véve azonos – a lemezen tárolt fájlok nevét és helyét tárolják – jellegzetességeik azonban eltérőek. A gyökérkatalógus – eddig egyszerűen katalógusnak neveztük – mérete és helye a lemezen rögzített. Az alkatalógus a gyökér kibővítése, mérete nem rögzített, és a lemezen bárhol lehet. A DOS 2.00-s változata, vagy az annál újabbak használják.

Az alkatalógusok a lemez adatterületén találhatók, mint egy közönséges fájl. Akár egy közönséges fájl, az alkatalógus is korlátlanul növekedhet, amíg a lemez be nem telik. Minden típusú lemezen létrehozhatók és használhatók. Mivel azonban értékes adatterületet foglalnak el, elsősorban a merevlemeznél szokás használni őket; hajlékonylemeznél kerülni szokták.

Az alkatalógusok mindig egy szülő katalógusához kapcsolódnak; amely vagy gyökérkatalógus, vagy egy másik alkatalógus. Ez több szintet is alkothat, így fastruktúrát hozva létre.



A szülőkatalógusban egy bejegyzés tartozik minden alkatalógushoz, ez éppen olyan, mint bármely más fájlé (kivéve, hogy az attribútumbájt jelzi, hogy alkatalógusról van szó, és a méretét jelző bájt értéke 0). A tényleges méretet úgy lehet megállapítani, hogy végigkövetjük a FAT-ban a hozzá tartozó láncot.

Ha egy alkatalógust hoznak létre, két különleges bejegyzés rögtön belekerül, ezek a "." és a ".." fájlnevek. Ezek úgy viselkednek, mint újabb alkatalógusok bejegyzései, de a "." a jelenlegi alkatalógust jelenti, a ".." pedig a szülőkatalógust. Ezekben a kezdő klaszter száma ezen alkatalógus, ill. a szülőkatalógus kezdetét adja meg. Ha a kezdő klaszter száma 0, ez azt jelenti, hogy a szülő a gyökérkatalógus.

Ha egy fájl mérete csökken, akkor általában számítani lehet rá, hogy a DOS felszabadítja a nem használt területet. Alkatalógusok esetében azonban a bejegyzések törlésével szabaddá vált területeket nem szabadítja fel, míg a teljes alkatalógust nem törlik.

Szeretnénk hangsúlyozni, hogy ha új elemre van szükség a katalógusban, a DOS az első szabad elemet felhasználja, azaz újra felhasználja egy törölt fájl régi elemét, ezzel az információ visszanyerését lehetetlenné teszi.

A fájl nevének első bájtján előforduló harmadik különleges kód a "." (hexa 2E), ez alkatalógust jelez. Ha a második bájt tartalma is 2E, akkor a jelenlegi alkatalógus szülőjéhez tartozó bejegyzéssel van dolgunk. Ekkor a kezdő klaszterszám (7. mező) a szülőkatalógus klaszterszámát tartalmazza.

A 2. mező: a fájlnév kiterjesztése

A fájl nevét közvetlenül követi a fájlnév kiterjesztése, ASCII-kódban. Hossza 3 bájt, és a fájl nevéhez hasonlóan jobbról szóközökkel egészül ki, ha 3 karakternél rövidebb. Míg a fájl nevéhez legalább egy érvényes karaktert tartalmaznia kell, addig a kiegészítése lehet üres is. Általában a fájlnév kiegészítésére ugyanazok a szabályok vonatkoznak, mint a fájlnévre.

📖 MEGJEGYZÉS: ha a katalógusban lemezcímke bejegyzés található, akkor a fájlnév és a kiegészítés mezejét egyetlen összevont 11 bájtos mezőként értelmezzük. Ennél a közbeékelt szóközök megengedettek. Általában nem szokás kisbetűt használni a címkében, bár használata nem tilos.

A 3. mező: a fájl attribútumai

A katalóguselem 3. mezeje 1 bájt hosszú. Ennek minden bitje a bejegyzés típusát írja le. Az attribútumbájt minden bitjének külön jelentése van. (Ez látható az 5.8. táblázatban.)

A 0. bit, a legmagasabb helyiértékű bit, azt jelzi, hogy a fájl csak olvasható. Ebben az állapotban a fájlt DOS-művelettel nem lehet megváltoztatni vagy törölni. Hangsúlyozni kell, hogy a DOS 1 változatok figyelmen kívül hagyják ezt a bitet, mert fontos ugyan az adatvédelem szempontjából, de korántsem biztos.

Az 1. és a 2. bit azt jelzi, hogy egy fájl rejtett- vagy rendszerfájl. A rejtettként, rendszerként vagy egyszerre mindkettőnek jelzett fájlok közönséges DOS-műveletek számára (**DIR**) láthatatlanok. Programunkkal úgy férhetünk hozzá ilyen fájlokhoz, hogy a fájl kontrollblokkban (FCB) átállítjuk ezt a két bitet. (☛ L. a 16. fejezetet.) A DOS IBMBIO.COM és IBMDOS.COM fájljai – néha IO.SYS és MSDOS.SYS néven találkozunk velük – egyszerre rejtett- és rendszerfájlok. A rendszerattribútumnak semmi jelentősége sincs: csak azért hagyták meg, hogy átmentsék a CP/M egyik jellegzetességét, aminek semmi köze a DOS-hoz.

A 3. bit azt jelzi, hogy a bejegyzés címke, vagyis itt a lemezt azonosító név található. A címke csak a gyökérkatalógusban fordulhat elő. A 8 mezőnél ez a bejegyzés csak néhányat használ. Maga a címke az egyesített fájlnév és a fájlnév-kiegészítés mezőben található, ebből

5.8. táblázat
Az állománytattribútum 8 bitje

Bit 7 6 5 4 3 2 1 0	Értéke: Decimális	Értéke: Hexa-decimális	Jelentése
. 1	1	1	Csak olvasható
. 1 .	2	2	Rejtett
. 1 . .	4	4	Rendszer
. . . . 1 . . .	8	8	Lemezcímke
. . . 1	16	10	Alkatalógus
. . 1	32	20	Archiválás
. 1	64	40	Nincs felhasználva
1	128	80	Nincs felhasználva

a szempontból ez egy mezőnek minősül. A méret- és a kezdő klasztermező kihasználatlan, de kitöltött a dátum és az idő.

A 4. bit az alkatalógus jelzője. Ez katalógusbejegyzésre utal, ami viszont további alkatalógusokat tartalmaz. Mivel az alkatalógust közönséges adatfájlként tárolja a lemez, szükség van egy kiegészítő bejegyzésre. Ennél a bejegyzésnél a méretmező kivételével a többi ki van használva – a méretmező értéke 0. Az alkatalógus tényleges méretét úgy lehet megállapítani, hogy a FAT-ban végigkövetjük a helyét kijelölő láncot.

Az 5. bit az ún. archiválási bit. Ez a merevlemezen tárolható sok-sok fájl kimentését támogatja. A bit értéke 0 azoknál a fájloknál, amelyek tartalma nem változott az utolsó kimentés óta. Közönséges esetben a bit értéke a hajlékonylemez fájljainál 1. Hajlékonylemezeknél ennek a bitnek nincs semmi hasznos funkciója.

A 4. mező: fenntartott

Ezt a 10 bájtos mezőt későbbi felhasználás céljára tartják fenn. Általában mind a 10 bájt értéke hexa 00.

Az 5. mező: az idő

Az 5. mező 2 bájton tartalmazza azt az időpontot, amikor a fájlt létrehozták, vagy utoljára használták. A dátummezővel együtt használják; a kettőt együtt mint egy 4 bájtos, előjeltelen számot lehet kezelni. Ezt a 4 bájtos egészet össze lehet hasonlítani más katalóguselemek megfelelő mezejével, hogy megállapítsuk: melyik nagyobb vagy kisebb, esetleg egyenlők-e. Magát az időt előjeltelen szavas egészként kezelhetjük, és a következő formulával számítjuk ki az órából, a percből és a másodpercből:

$$\text{Idő} = \text{Óra} \times 2048 + \text{Perc} \times 32 + \text{Másodperc} \div 2$$

Az óránál 24 órát vettek figyelembe, értéke 0 és 23 között lehet. Mivel az idő tárolására használt 2 bájtos szó 1 bittel rövidebb annál, ami a másodpercek tárolásához kellene; a másodpercet 2 másodperces egységekben tárolják; értéke 0 és 29 között lehet. Az itt tárolt 5-ös érték 10 másodpercet jelent. A 11 : 32 : 10 időpontot 23 557 értékkel ábrázolja.

A 6. mező: a dátum

A 6. mező a létrehozás vagy az utolsó változtatás dátumát tárolja. Az időmezővel együtt használják, a kettőt együtt mint egy 4 bájtos előjeltelen számot lehet kezelni. Ezt a 4 bájtos egészet össze lehet hasonlítani más katalóguselemek hasonló mezejével, hogy megállapítsuk: melyik nagyobb vagy kisebb, esetleg egyenlők-e. Magát a dátumot előjeltelen szavas egészként kezeljük, és a következő formulával számítjuk ki az évből, a hónapból és a napból:

$$\text{Dátum} = (\text{Év} - 1980) \times 512 + \text{Hónap} \times 32 + \text{Nap}$$

Megfigyelhetjük, hogy a formula az évet tömöríti azzal, hogy 1980-at levon belőle. Így az 1984-es évből 4 lesz. A formula 1984. december 12-ét 2828-nak ábrázolja:

$$(1984 - 1980) \times 512 + 12 \times 32 + 12 = 2828$$

Bár ezzel a képlettel 2108-ig tudunk dátumot tárolni, a DOS csak 2099-ig tartó naptárt használ.

A 7. mező: a kezdő klaszter sorszáma

A 7. mező egy 2 bájtos érték, azt adja meg, hogy hányas klaszteren kezdődik a fájl adatterülete. Ez a belépési pont a fájl részére lekötött helyet mutató láncba, amit a FAT tárol. Azoknál a fájloknál, amelyek nem foglalnak el helyet, valamint a lemezt azonosító kötetcímke bejegyzéseknél a kezdő klaszter száma 0, nem pedig a FAT által a fájl vége jelzésére használt hexa FFF érték.

A 8. mező: a fájl mérete

A katalóguselem utolsó mezeje megadja, hogy a fájl mérete hány bájt. A méret 4 bájtos előjeltelen egészként van kódolva, ez olyan nagy méretű fájlokat enged meg, ami jóval meghaladja lemezeink kapacitását.

A DOS számára ez a mező a fájl pontos méretét jelzi. Néha azonban ez az érték nagyobb lehet a tényleges méretnél. Egyes szövegszerkesztők pl. az általuk létrehozott ASCII kódú szövegfájlok valódi végét a *CTRL-Z*-karakterrel – **CHR$(26)**, hexa 1A – jelzik. Ezeknél a fájloknál a méretmező a valóságosnál nagyobb számot is tartalmazhat, pl. a 128 bájt legközelebbi egész számú többszörösére felkerekített értéket. A szövegszerkesztőknél ez gyakran előfordul, mivel ezek az adatokat nagyobb blokkokban olvassák, nem pedig bájtonként. Lényeges tudni, hogy DOS-szal olvasott fájlnál akkor kapunk fájl vége jelzést, ha eléri a fájl mérete által jelzett határt, vagy ha a FAT-ban a lekötött területet mutató lánc végére ér – ezt hexa FFF jelzi – (attól függ, melyik jön előbb).

Az adatok területe

Az összes adatfájl és az alkatalógusok – amelyek az adatfájlokhoz nagyon hasonlóan viselkednek – a lemez utolsó és legnagyobb területén találhatók.

A fájlok igény szerint kapnak tárolóterületet, egyszerre egy klaszternyit. (Emlékeztetünk rá, hogy egy klaszter, egy vagy több szomszédos szektorból áll; a szektorok számát egy klaszterben a lemez formátuma rögzíti.) Ha egy fájlt létrehozunk, vagy egy létező fájlt bővítünk, nő a fájl által lekötött terület. Amikor újabb helyre van szükség, egy klasztert kap a fájl. A DOS 1-es és 2-es változatainál mindig az első szabad klasztert köti le a fájl. A DOS újabb változatai bonyolultabb szabályok szerint választják ki a következő klasztert (ezt most nem részletezzük).

Sokszor előfordul, hogy egy fájl folyamatos tárolóterületet foglal el. Az is megtörténhet azonban, hogy nem szomszédos blokkokra kerül. Különösen akkor, ha egy létező fájlt bővítünk, vagy új fájlt egy törölt fájl után felszabadult területen hozunk létre. Egyáltalán nem szokatlan, hogy egy fájl adatai az egész lemezen szétszórtan helyezkednek el.

Az ilyen jellegű széttördelés bizonyos esetekben lassítja a fájl adatainak elérését. Ezenkívül sokkal nehezebb egy véletlenül kitörölt fájlt regenerálni, ha az szét van tördelve, mert nehezebb megtalálni a fájlhoz tartozó szektorokat. Más hatása viszont a széttöredezésnek nincs. Általában a programoknak nem kell azzal törődni, hogy hol találhatók adataik a lemezen. Két módon is ellenőrizhetjük, hogy egy fájl széttöredezett-e: használhatjuk a **CHKDSK**-parancs **/V** opcióját, vagy egy Norton Utilities-hez hasonló programmal grafikus térképet készíthetünk a lemez egyes fájljainak elhelyezkedéséről.

Ha a hajlékonylemez fájljai válnak széttöredezettekké, ezt úgy szüntethetjük meg, hogy egy frissen formattált lemezre átmásoljuk őket. Természetesen a fájl ismét széttöredezhet, ha

gyakran kell a tartalmát módosítani. Merevlemezen nehéz bármit is tenni a széttöredezés elhárítására, nem is kell túl sokat törődni vele. Azért mégis legyen tudomásunk róla, bár egy széttöredezett fájl gyakorlatilag semmi veszélyt sem jelent.

Attól függetlenül, hogy megnézi-e valaki a széttöredezett fájlokat, a fogalom ismeretében könnyebb megérteni, hogyan használja a DOS a fájl allokáció táblázatát, a FAT-ot; hogyan biztosít helyet a fájloknak, és hogyan képezi a FAT-ban a fájlt alkotó klasztereket összekapcsoló allokációs láncot.

A fájl allokáció táblázata

A FAT rögzíti a lemez helykihasználását. Elemezni fogjuk a FAT szerkezetét (ez viszonylag egyszerű és kézenfekvő) és tárolási módját (ez valamivel bonyolultabb).

Mint korábban említettük, a szabványos lemezes formátumok 2 példányban tárolják a FAT-ot, de lehet ennél több vagy kevesebb, akár 1 példány is. A FAT egy-egy példánya 1 szektort foglal el a nyolcszektoros lemezen és 2 szektort a kilencszektoroson. A QD – 15-nek nevezett nagykapacitású formánál a FAT 7 szektort foglal el.

A legtöbb lemezes formánál a DOS 2 példányban tárolja a FAT-ot (hátha az egyik megsérül, vagy olvashatatlanná válik), de nincs tudomásunk arról, hogy másra is felhasználná ezt a redundanciát. A CHKDSK-program (amely a FAT és a katalógus legtöbb lehetséges hibáját ellenőrzi) tudomást sem vesz róla, ha a két FAT eltérő.

A FAT-nak két formátuma van: a 12 bites és a 16 bites. A 12 bites FAT az általánosabb és egyben a bonyolultabb. A 16 bites FAT-ot csak olyan lemeznél használják, ahol a lemez kapacitásához a 12 bites FAT nem elég (pl. az AT 20 Mbájt-os merevlemezénél). Először a szabványos 12 bites formát ismertetjük, majd elmondjuk, hogy a 16 bites FAT ettől miben különbözik.

A FAT legföljebb 4086 számot tartalmazó táblázat. Az egyes számok 0 és 4095 (hexa FFF) közé esnek. Az adatterület minden klaszteréhez egy bejegyzés tartozik. Az egyes elemekben álló szám az illető elemnek megfelelő klaszter állapotát és felhasználását jelzi. Figyeljük meg, hogy a FAT táblázatban a számok 3 hexadecimális számjegynél nem hosszabbak. Ez kulcsfontosságú a 12 bites FAT tárolása szempontjából (amint ezt hamarosan látni fogjuk).

Ha egy FAT-elem értéke 0, ez azt jelenti, hogy az illető klaszter szabadon felhasználható. Ha egy FAT-elem értéke 4087 (hexa FF7), és a FAT-elem nem része egyik allokációs láncnak sem, akkor a megfelelő klaszter formattálási hiba miatt használhatatlan. Ezt nevezik a hibás sáv megjelölésének is.

🕮 MEGJEGYZÉS: Érdemes egy pillanatra elidőzni annál a kérdésnél, hogy miért nincs abban semmi riasztó, ha egy lemezen – főleg merevlemezen – „hibás sáv" jelzések vannak. Valójában teljesen megszokott, hogy egy merevlemezen van néhány hibás folt. Annak az AT-nek a merevlemezén, amin ez a könyv készült 3 (kisebb) hibás sávokat tartalmazó terület van. A lemezt formattáló program észleli a hibás sávokat, és ennek megfelelő jelzést tesz a FAT-ba (ezt az imént megbeszéltük). A későbbiekben a hibás sáv jele tudatja a DOS-szal, hogy ezeket a területeket ki kell hagyni. A hibás sáv a hajlékonylemeznél is megszokott jelenség, de ellentétben a merevlemezzel, egy hajlékonylemezt el lehet dobni, és csak a tökéletest kell használni.

A klaszterek számozása 2-től kezdődik, és az utolsó szám eggyel nagyobb a lemez klasztereinek számánál. (L. az 5.9. táblázatot.) Ha a 12 bites FAT valamelyik elemében 2 és 4080 (hexa 02 és FF0) közötti szám található, ez azt jelenti, hogy az illető klasztert egy fájl használja.

5.9. táblázat

A klaszterek száma a különböző DOS-formátumokban

Formátum	Szektorok	Szektor klaszterenként	Klaszterek	A klaszterszámok tartománya
S-8	313	1	313	2-314
D-8	630	2	315	2-316
S-9	351	1	351	2-352
D-9	708	2	354	2-355
QD-9	1422	2	711	2-712
QD-15	2371	1	2371	2-2372

Ha a FAT elem értéke 4095 (hexa FFF), ez azt jelenti, hogy ebben a klaszterben található a fájl adatainak vége. A 4088 és 4094 (hexa FF8 és FFE) közötti értékek felhasználása hasonló lehetne, de tapasztalatunk szerint ez mégsincs így.

Láthattuk, hogy a FAT elemei a terület kiosztását – allokálását – lánccal ábrázolják. Ebben a láncban a katalógusbejegyzés tartalmazza a kezdő klaszter sorszámát (☛ l. a 7. mező leírását!), a FAT bejegyzései mutatják a következő klasztert és a fájl végét is. (L. az 5.10. táblázatot.) Amikor egy fájlt törölnek, akkor a lekötött területet jelző FAT-elemekbe a szabad jelzés kerül – 0 értéket kap – a fájl tényleges tartalma azonban az adatterületen nem változik, megmarad továbbá a fájlhoz tartozó katalógusbejegyzésben is az információ nagy része.

5.10. táblázat

Egy állomány helyfoglalási lánca a FAT-ban

Fat-bejegyzés		Értéke		Jelentése
		Decimális	Hexadecimális	
	0	253	FD	A lemez kétoldalas, dupla sűrűségű
	1	4094	FFE	Fel nem használt bejegyzés; nem használható
A katalógus-bejegyzésből; a fájlhoz tartozó lánc kezdete	2	3	3	Az állomány következő klasztere a 3-as
	3	5	5	Az állomány következő klasztere 5-ös
	4	4087	FF7	Használhatatlan klaszter: hibás sáv
	5	6	6	Az állomány következő klasztere 6-os
	6	4095	FFF	Az állomány utolsó klasztere, egyben az állomány allokációs láncának vége
	7	0	0	Fel nem használt bejegyzés: szabad

A FAT szerkezete numerikus értékek meglehetősen egyszerű táblázata, tárolásmódja – a tömörség érdekében – elég bonyolult. A 8088-as adatformáit kihasználva alkalmazza a szavak fordított sorrendű tárolását. A FAT-nál az egyszerűséget feláldozták a hatékonyság rovására.

A klaszterszámok tartományát úgy definiálták, hogy 4095-öt (hexa FFF) ne haladják meg. Ez lehetővé teszi, hogy a 3 hexadecimális számjegyet 12 biten, azaz másfél bájton lehessen tárolni. A FAT elemei párokba vannak szervezve, egy pár 3 bájtot foglal el (0 és 1 foglalja az első 3 bájtot, 2 és 3 a következő hármat és így tovább). A három bájtot így kell dekódolni: ha egy FAT-bejegyzés páros értéke 123 és 456, akkor az őket tartalmazó 3 bájt így néz ki hexadecimálisan: 23 61 45. A fordított eljárás így fest: ha a 3 bájtban AB CD EF áll, akkor a 2 FAT-érték DAB és EFC. Az 5.11. táblázatban látható az S – 8, S – 9 és D – 8 formátumnál az utolsó klaszterszám páros, ezért a FAT-ban egy mesterséges (jelentés nélküli) értékkel egészül ki.

5.11. táblázat
A szokásos lemezes formátumok azonosítóbájtjainak értékei

Formátum	Azonosítóbájt
D – 8	FF
S – 8	FE
D – 9	FD
S – 9	FC
QD – 9	F9
QD – 15	F9

A kódolás különösnek tűnik a mi gondolkodásunk számára, gépi kódú utasításokkal viszont gyorsan és hatékonyan elvégezhető. Ha adott egy klaszter száma, akkor a FAT-értéket úgy kapjuk meg, hogy a klaszterszámot 3-mal megszorozzuk, 2-vel elosztjuk, vesszük az eredmény egész részét, és ezt a FAT-on belül relatív címként használjuk. Ha kivesszük az így kapott címen kezdődő szót, akkor megvan a FAT-bejegyzés 3 hexa számjegye és egy fölösleges számjegy, amitől többféle gépi kódú utasítással is megszabadulhatunk. Ha a klaszter száma páros, akkor a legmagasabb helyértékű számjegyet hagyjuk el; ha páratlan, akkor az alacsony helyértékűt. Az így kapott érték a fájl következő klaszterének száma, kivéve az FFF értéket, az a fájl végét jelzi.

Ezt a bonyolult sémát eredetileg nyolcszektoros hajlékonylemezre dolgozták ki. Korántsem annyira ideális, ha más formátumnál használják (pl. kilencszektorosra), ahol a FAT egy szektornál kicsit nagyobb lesz. Mindent összevetve, mégis tömör és hatékony ez a módszer.

Eddig a 12 bites FAT részleteit vizsgáltuk. Ez 4080 klaszter tárolására alkalmas. Ha egy lemezes formátumban ennél több klaszter van, akkor a 16 bites FAT-ra van szükség.

AT

A 16 bites FAT pontosan úgy működik, mint a 12 bites, de egyszerűbb. A 16 bites FAT elemei 4 bittel hosszabbak, ez megnöveli a klaszterszámok tartományát. Mivel a 16 bit éppen 2 bájt (azaz 1 szó), a 16 bites FAT-nál nincs szükség a 12 bites FAT bonyolult elrendezésére. Ehelyett a 16 bites FAT szavas értékek egyszerű táblázata, egyik szó következik a másik után.

A 16 bites FAT különleges értékei (pl. a hibás sáv jelzésére használt érték) egyszerűen a 12 bites FAT-nál használt érték logikus kiterjesztése: egy F hexadecimális számjegy jelenik meg a legmagasabb helyértékű számjegy helyén. A fájl végét például a hexadecimális FFFF (FFF helyett) a hibás klasztert FFF7 (FF7 helyett) jelzi.

Mint már említettük, az igazi adatklaszterek számozása 2-vel, a FAT számozása pedig a 0-s és az 1-es elemmel kezdődik. Ezt az első 2 FAT-elemet sem a 12 bites, sem a 16 bites FAT-ban nem használjuk klaszterek státuszának jelzésére (fenntartottak), így a FAT legelső bájtját a lemez formátumának azonosítójaként lehet felhasználni. Nem szabad azonban azt képzelni, hogy ezek az azonosítók egyértelműen azonosítják a formátumot, ez nem feltétlenül igaz. Ha végignézzük az összes használt lemezformátumot, jó néhány átfedést találunk. Legyünk óvatosak!

Egy program megvizsgálhatja a lemez formátumát a FAT azonosító bájtjának (ID) kiolvasásával. A formátum megállapításának „hivatalos" módja a DOS 27-es függvénye (hexa 1B). (☛ Ezt bővebben a 16. fejezetben ismertetjük.)

Néhány megjegyzés a FAT-tal kapcsolatban

Általában egy program nem szokta a FAT-ot vizsgálni vagy változtatni: a FAT teljesen a DOS hatáskörébe tartozik. Ez alól csak azok a programok jelentenek kivételt, amelyek a DOS-szal nem támogatott helykiosztási feladatokat hajtanak végre. Ilyen a törölt fájlokat helyreállító program (pl. a Norton Utilities programcsomag UnErase programja).

Fontos tudni, hogy a FAT logikai szerkezete károsodhat: lehetséges ciklikus allokációs lánc, amely a lánc egy korábbi elemére mutat vissza; lehetséges, hogy két lánc ugyanarra a klaszterre mutat; egy klaszter izolálódhat, vagyis jelölés szerint a klaszter használatban van, de egyetlen allokációs láncnak sem része; előfordulhat, hogy a fájl vége jel (a hexa FFF vagy FFFF) hiányzik. A DOS CHKDSK és RECOVER nevű programjai arra valók, hogy az ilyen jellegű hibák legtöbbjét feltárják és kijavítsák, már amennyire ez egyáltalán lehetséges.

(☛ A FAT allokációs láncainak és a DOS által nyilvántartott fájlméretnek a kölcsönös viszonyáról további megjegyzések találhatók e fejezet elején, a katalógusnál.)

Megjegyzések

Bár ebben a fejezetben részletes adatokat közöltünk a lemez közvetlen használatáról, beleértve a betöltőrekordot, a FAT-ot és a katalógusokat; nem lenne szerencsés ezeket direkt módon használni, hacsak nincs valami kényszerítő ok rá (pl. másolás ellen védő programoknál). Általában tartsuk szem előtt a műveletek szabványos hierarchiáját, és az igényeinknek megfelelő legmagasabb szintű szolgáltatást használjuk:

- Az első lehetőség: a nyelvi szolgáltatások. Ezek a programozási nyelvekbe épített lehetőségek. Pl. a BASIC nyelv **OPEN**- és **CLOSE**-utasítása.
- A második lehetőség: a DOS szolgáltatásai. Ezeket a 15. és 16. fejezet ismerteti.
- A harmadik lehetőség: a ROM-BIOS lemezes szolgáltatásai. Ezek a 10. fejezetben találhatók.
- A negyedik lehetőség: direkt parancsokkal. Pl. a hajlékonylemezes vezérlő, az FDC programozása a portokon keresztül kiadott parancsokkal.

A PC-család legtöbb lemezes műveletét könnyen végre lehet hajtani a programozási nyelv nyújtotta lehetőségekkel. Van azonban két nyilvánvaló helyzet, amikor szokatlanabb meg-

oldáshoz kell folyamodni. Az egyiket már említettük: a DOS-szal egy szinten levő lemezes műveleteket kell végrehajtani. Ezt akkor kell igénybe venni, ha a DOS CHKDSK-jához vagy a Norton Utilitites-hez hasonló programot kell írni. A másik eset a másolás elleni védelem. Valamilyen értelemben az összes másolás ellen védő rendszer tartalmaz valamilyen nem szokványos lemezes I/O műveletet. Ez általában a ROM-BIOS szolgáltatásainak igénybevételéhez vezet, de felvetheti annak a szükségességét is, hogy közvetlenül kelljen programozni a hajlékonylemezes vezérlőt.

Másolás elleni védelem

A másolás ellen védő rendszerekből jó néhány van kereskedelmi forgalomban, köztük egy a szerző Access Tools for the IBM PC készletében. Ez utóbbi nem túl biztonságos. Előfordulhat, hogy valaki mégis maga akarja a sajátját elkészíteni.

Igen sokféle módon lehet a másolás elleni védelemhez közelíteni. Talán a legközönségesebb eljárás az, hogy a ROM-BIOS formattáló szubrutinjával újra formázzuk egyes sávok szektorait. Mivel a DOS nem tudja elolvasni azokat a szektorokat, amelyek eltérnek a neki megfelelő formátumtól, a DOS COPY-programjával nem lehet olyan lemezt másolni, amelyen a szektorok között alkalmanként furcsa szektorok fordulnak elő. Természetesen a DOS-nak ez a korlátja több céget is arra késztetett, hogy olyan másoló programot készítsen, amely tetszőleges méretű szektorokat elolvas és lemásol, ez tehát nem nyújt igazán hatékony védelmet a másolás ellen.

A következő szinten, két különleges dolgot érdemes megemlíteni a másolás elleni védelemmel kapcsolatban. *Először:* a legagyafúrtabb és legmegfejthetetlenebb védelmi sémák a hajlékonylemezes vezérlő, az FDC nem dokumentált lehetőségeinek felfedezésén alapszanak. *Másodszor:* egyes sémák akarva vagy akaratlanul a különböző hajlékonylemezes meghajtók különleges tulajdonságain alapulnak. Előfordul az is, hogy a meghajtó szoftvervezérlésének részleteire támaszkodnak. Ez a PC-családon belül eltérő lehet. A PCjr meghajtót vezérlő szoftvere pl. a PC-től és az XT-től nagyon eltérő módon működik. Az AT szoftvere megint más. Előfordulhat tehát, hogy a másolás ellen védett program az egyik modellen működik, a másikon pedig nem – annak ellenére, hogy a védelmi rendszerhez hozzá sem nyúltak. Ha másolás elleni védelemmel találkozunk, erre mindig figyeljünk.

Konkrétabb alapelvekkel itt nem szolgálhatunk, mindössze arra hívjuk fel a figyelmet, hogy a sikeres védelmi séma kulcsa a változatosság és a találékonyság.

6 A billentyűzet alapjai

Bevezetés • 110

A billentyűzet működése • 110

Kommunikáció a ROM-BIOS-szal • 111
A letapogatási kód átalakítása • 112
ASCII-kódok közvetlen bevitele • 114

A billentyűzet adatformái • 114

Az ASCII-billentyűk • 114
A különleges billentyűk • 115

A billentyűzet vezérlése • 117

A státuszbájtok • 117

Megjegyzések • 119

A PCjr sajátosságai • 120

A PCjr billentyű műveletei • 121
A PCjr-billentyű státuszbájtja • 121

Az AT sajátosságai • 123

Bevezetés

Ebben a fejezetben elsősorban a szabványos IBM PC billentyűzetével foglalkozunk, de szólunk a PCjr és az AT kissé eltérő billentyűzetéről is. Kerülni fogjuk az IBM PC különleges modelljeinek (a 3270 PC-nek, a PCjr-nek és az AT-nek), valamint a család nem IBM által gyártott tagjainak a részletes tárgyalását, mert billentyűzetük nem pontosan egyezik a szabványos PC-jével. Ezek a szabványtól eltérő billentyűzetek legtöbbször a PC-szabvány erősen kibővített (vagy éppen csökkentett) képességű változatai. Az AT-n pl. az IBM áthelyezett néhány billentyűt, bevezetett egy új billentyűt is, divatosabb áramköri elemeket alkalmazott; de szerencsére nem sokat változtatott a működési paramétereken. A PCjr-nek a szabványos PC billentyűzeténél kevesebb billentyűje van, mégis meggyőzően utánozza az eredetit. Szerencsére a bővebb PC-család tagjainál is általános gyakorlat, hogy billentyűzetük megegyezik az eredetivel, vagy jól utánozza azt. Ezek a kis különbségek azonban a programozóknak nem jelentenek gondot.

A fejezet első részében arról lesz szó, milyen a billentyűzet és a számítógép kölcsönhatása a hardver és a szoftver szintjén. A második részben azt nézzük meg, hogyan kezeli a ROM-BIOS a billentyűkről jövő információt, és hogyan adja át a programoknak. Aki egy kicsit játszani akar a billentyűzet vezérlésével, vegye figyelembe megjegyzéseinket is, és ne a fejezetben található információt alkalmazza, hacsak nincs rá különös oka. Olyan, mint amikor a billentyűzet működését akarja módosítani a népszerű Prokey billentyűzetjavító program. (☛ Ha valaki ilyen terveket forgat a fejében, vessen egy pillantást a 11. fejezetben található ROM-BIOS-billentyűzetszolgáltatásokra.)

A billentyűzet működése

A PC billentyűzetében egy 8048-as vezérlőt találunk. Ez sokféle, a rendszer adminisztrációs terheit csökkentő, feladatot lát el. A 8048 legfontosabb feladata a billentyűk figyelése, és a billentyűk lenyomásának vagy elengedésének jelzése a ROM-BIOS számára. Ha bármelyik billentyű 0,5 másodpercnél hosszabb ideig van lenyomva, akkor a 8048 megadott időközönként ismétlőjelet küld. A 8048 korlátozott hibamegállapításra és hibaellenőrzésre is képes; ezenkívül találunk benne egy puffert, amely 20 billentyűműveletet tárolhat arra az esetre, ha a számítógép nem tudná azokat fogadni (ez azonban ritkán fordul elő). [AT] Az AT-ben 8042-es billentyűzetvezérlő IC van, de lényegében ez is a 8048-hoz hasonló feladatokat lát el.

Billentyűzetjavító programok

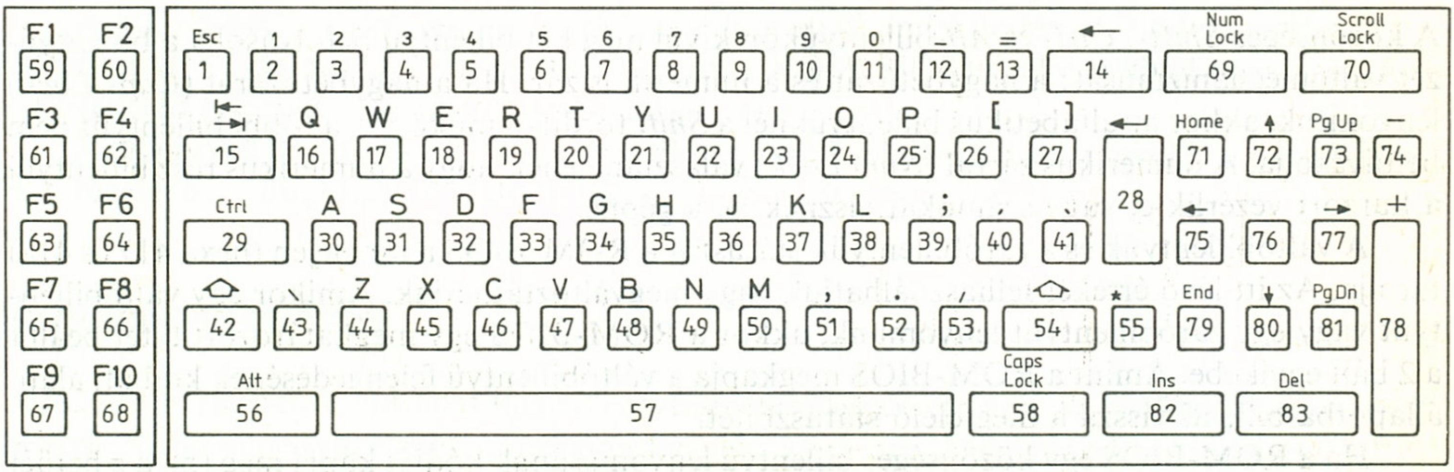
A PC rugalmasan tervezett szoftverének köszönhetően létre lehet hozni a billentyűzetet javító programokat. Sok ilyen program van forgalomban, de valószínűleg a Prokey a legismertebb és a legnépszerűbb.

A billentyűzetjavító programok figyelemmel kísérik a billentyűkről bejövő adatot, és kívánságunknak megfelelően megváltoztatják azt. Általában ezeket a programokat billentyűzetmakróknak nevezett utasításokkal értesítjük arról, hogy milyen billentyűk leütését figyeljék, és mit változtassanak meg. A változtatás lehet egy leütött billentyű figyelmen kívül hagyása (mintha a billentyűt le sem ütötték volna), egy billentyű leütésének helyettesítése egy másikkal, vagy egy billentyű lenyomásának átalakítása billentyűlenyomás-sorozattá. A billentyűmakrót legtöbbször a gyakran használt kifejezések lerövidítésére használjuk. Utasíthatjuk pl. a Krokey-t, hogy az *Alt-s* billentyűkombináció lenyomására írja

ki a levelezésben használt formulát, a „Szívélyes üdvözlettel”-t. Arra is felhasználhatjuk a makrókat, hogy a programutasításokat lerövidítsük. Így a két-három billentyűvel leírható parancsot egyetlen billentyű lenyomásával pótolhatjuk.

A billentyűzetjavítók működése két szolgáltatás jó tulajdonságainak kombinációján alapszik: az egyik a DOS, a másik a PC ROM-BIOS része. A DOS azt is lehetővé teszi, hogy a javítóprogram állandóan figyelje a gép működését, a vezérlést átengedve egy hagyományos programnak (pl. egy szövegszerkesztőnek). A ROM-BIOS azt biztosítja, hogy a billentyűzetről származó adatokat egy program eltérítse, megvizsgálja és megváltoztassa – mielőtt továbbítaná. Egy Prokey-hez hasonló program, a DOS „rezidenssé tevő” szolgáltatását felhasználva a tárban aktív marad, mialatt más program fut. Ezután a ROM-BIOS által nyújtott billentyűzet megfigyelésének lehetőségét is kihasználva, előzetesen megvizsgálja, és szükség szerint megváltoztatja az adatokat.

Valahányszor lenyomunk vagy elengedünk egy billentyűt a PC-n, a billentyűzet áramkörei egy 1 bájtos kódot állítanak elő. Ez a letapogatási kód egyértelműen azonosítja a történeteket. A billentyűzet más kódot állít elő egy billentyű lenyomásakor, ill. felengedésekor. Lenyomáskor a letapogatási kód 1 és 83 közé esik – egy szabványos PC billentyűzetén. Felengedéskor 128-cal (hexa 80) nagyobb értéket kapunk, mint a billentyű lenyomásakor: ez a 7. bit 1-re állítását jelenti. Ha pl. a *Z* betűt ütjük le, akkor a billentyűzet a 44-es kódot állítja elő; ha viszont felengedjük, akkor a 172-est (44 + 128). (A 6.1. ábrán láthatók a szabványos billentyűzet és a hozzájuk tartozó letapogatási kódok.)



6.1. ábra. A szabványos PC billentyűzete és a letapogatási kódok

Gépeléskor a billentyűzet nem ismeri a lenyomott billentyűk jelentését; csupán jelenti, hogy pontosan mi történt. A ROM-BIOS billentyűzet szubrutinjainak a feladata, hogy a program által is használható értelmes információvá alakítsák át a billentyűk egyszerű lenyomását. A billentyűzet portok és megszakítások segítségével kommunikál a ROM-BIOS-szal.

Kommunikáció a ROM-BIOS-szal

Valahányszor a PC billentyűzetén egy billentyűt lenyomunk vagy felengedünk, a PC erről a 9-es megszakításon – az ún. billentyűzetesemény megszakításon – keresztül értesül. Válaszul a 9-es megszakítás meghív egy megszakításkezelő szubrutint, amelyik a 96. (hexa 60) port beolvasásával megállapítja, mi történt pontosan. Ezután a várakozó letapogatási kód a BIOS-ba kerül, ahol a billentyűzetet kiszolgáló szubrutin 2 bájtos kóddá alakítja át. A 2 bájtos kódból az alacsonyabb helyiértékű (rendszerint a billentyűnek megfelelő) ASCII-kódot, a másik pedig

a letapogatási kódot tartalmazza. Különleges billentyűk (pl. funkcióbillentyűk vagy a numerikus rész billentyűi) esetén az alacsony helyiértékű bájtban 0 van, a másikban pedig a letapogatási kód. (☛ Erről részletesen a későbbiekben lesz szó.)

A BIOS-szubrutinok végül egy sorba állítják az átalakított kódot. A sor a tár alsó részén, a 0000 : 041E címen van. A kódok mindaddig itt maradnak, míg a billentyűbemenetre várakozó program (a DOS vagy a BASIC) nem kéri ki.

A letapogatási kód átalakítása

A letapogatási kód átalakítása kissé bonyolult, mivel az IBM billentyűzetein különböző váltó-billentyűkkel (*Shift*) módosítható a lenyomott billentyű jelentése. Ha a *Shift*-billentyűt és a *c*-billentyűt nyomjuk le, akkor *C* lesz az eredmény; ha a *Ctrl*-billentyűt és a *c*-billentyűt, akkor *Ctrl-C* vagy „állj" (break) jelzést állítunk elő. Mindkét példa a váltóbillentyűk különböző állását szemléltette. A váltó állapotát gépelés közben a *Shift*-, az *Alt*- vagy a *Ctrl*-billentyűvel változtathatjuk meg. Ameddig ezek közül valamelyiket lenyomva tartjuk, addig a ROM-BIOS tudja, hogy a következő lenyomott billentyűk jelentését a váltók befolyásolják.

A váltóbillentyűk és a záróbillentyűk

A közönséges *Shift-, Ctrl-* és *Alt*-billentyűkön kívül még két billentyű befolyásolja a billentyűzet váltómechanizmusát: a nagybetűzár és a numerikus zár. Ha a nagybetűzárat (*Caps Lock*) lenyomtuk, akkor az alfabetikus billentyűknél a *Shift* fordítva működik, a többi billentyűt nem befolyásolja. A numerikus zárral (*Num Lock*) választani lehet, hogy a numerikus rész billentyűi a kurzort vezérlik-e, vagy számokat visznek be a gépbe.

A váltóbillentyűk és a záróbillentyűk státuszát a ROM-BIOS a tár elején (hexa 417 és 418) tárolja. Az itt lévő értéket felhasználhatjuk vagy megváltoztathatjuk. Amikor egy váltóbillentyűt vagy egy záróbillentyűt lenyomunk, akkor a ROM-BIOS egy meghatározott bitet beállít a 2 bájt egyikébe. Amint a ROM-BIOS megkapja a váltóbillentyű felengedésének kódját, alapállapotba billenti vissza a megfelelő státuszbitet.

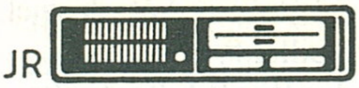
Ha a ROM-BIOS egy közönséges billentyű lenyomásának kódját kapja meg (pl. a *z* betűét vagy egy jobbra nyílét) először a váltó állapotát vizsgálja meg, azután állítja elő a billentyű 2 bájtos kódját. (☛ A státuszbájtot részletesen megbeszéljük a billentyűzet vezérlésének tárgyalásakor.)

Billentyűkombinációk

A ROM-BIOS a letapogatási kódok átalakítása közben folyamatosan figyel bizonyos billentyűzetkombinációkat: a *Ctrl-Alt-Del,* a *Shift-PrtSc*, a *Ctrl-Num Lock* és a *Ctrl-Break* kombinációt. E négy parancsszerű billentyűkombináció hatására a ROM-BIOS rögtön végrehajtja a megfelelő feladatot – ahelyett, hogy a pufferbe tenné a karaktereket.

A ***Ctrl-Alt-Del*** hatására az operációs rendszer újból betöltődik. Valószínűleg a *Ctrl-Alt-Del* a leggyakrabban használt billentyűkombináció. Mindaddig megbízhatóan működik, amíg a billentyűzetet kiszolgáló megszakítás is működik. Két oka lehet annak, hogy a billentyűzet megszakítása ne működjön: vagy a billentyűzet megszakítási vektora (hexa 36 – 39 tárcímek) változott meg, vagy volt egy megszakításokat letiltó **CLI** utasítás, de a megszakítást engedélye-

ző STI utasítás elmaradt. Mindkét esetben csak egy kiút van: a tápfeszültséget ki kell kapcsolni, néhány másodpercet várni, majd újra bekapcsolni; a bekapcsolási program az összes megszakítási vektort és szolgáltatást az eredeti állapotába állítja vissza.



MEGJEGYZÉS: Előfordulhat, hogy egy program hibásan, letiltva hagyja a megszakításokat. Ez a PCjr-nél lehetetlen, mert a billentyűzet a nemmaszkolható megszakítást – az NMI-t – használja.

A ***Shift-PrtSc*** a szabványos nyomtatóra írja a képernyő tartalmát. A művelet a BIOS alacsony szintjén zajlik, az 5-ös megszakításon keresztül. A nyomtató helyett a kiíratást úgy lehet más berendezésre átirányítani (nem ez a szokásos), hogy a *PrtSc* megszakítási vektorát átírjuk egy új szubrutinra. A DOS 2.00-nál és az új változatoknál a GRAPHICS.COM parancs kicselezi a *PrtSc* műveletet, mert először megvizsgálja az érvényben lévő megjelenítési módot. Ha grafikus mód van érvényben, akkor egy olyan szubrutin veszi át az irányítást, amelyik képpontként kiküldi a képet egy IBM-kompatíbilis grafikus nyomtatóra természetesen akkor, ha ez a géphez van kapcsolva. Különben a hagyományos képernyőnyomtatás lép működésbe, és az információt karakterenként fogja kiküldeni.

A ***Ctrl-Num Lock*** egy újabb billentyű lenyomásáig felfüggeszti a program futását.

A ***Ctrl-Break***re a számítógép „állj" (break) jelet generál a 27-es megszakítással. Ha programunkban átírtuk a 27-es megszakítás kezelőjét, akkor elfoghatjuk a jelet, és belátásunk szerint cselekedhetünk (esetleg figyelmen kívül is hagyhatjuk). Ha nem változtattuk meg a megszakítási szubrutint, akkor a DOS az eredeti szubrutint használja, és leállítja a programot.

A ROM-BIOS számára csak ezeknek a billentyűkombinációknak van jelentése. Ha a billentyűzetről értelmetlen kombinációt kap, azt a ROM-BIOS figyelmen kívül hagyja, és rátér a következő értelmes esemény feldolgozására.

A billentyűk kódolásának részletezése előtt szólunk a billentyűismétlésről és a két példányban meglevő – duplikált – billentyűkről.

A billentyűismétlés

A PC billentyűzetének automatikus billentyűismétlési képessége van, az IBM ezt *typematic*nek nevezi. A PC billentyűzetének áramkörei figyelik, hogy egy billentyűt mennyi ideig tartottuk lenyomva. Ha 0,5 másodpercnél tovább, akkor másodpercenként tízszer megkezdi a billentyű ismétlését úgy, hogy a billentyűlenyomás letapogatási kódját nem szakítja meg a felengedés kódja. Ebből a 9-es megszakítás intelligens kezelője el tudja dönteni, hogy tényleg megnyomták-e a billentyűket, vagy a typematic működik. A ROM-BIOS azonban nem mindig különbözteti meg a kettőt, billentyűkezelő szubrutinja minden egyes automatikusan ismételt billentyűt úgy tekint, mintha a billentyűt tényleg lenyomták volna, és ennek megfelelően értelmezi a billentyűt. Ha pl. az *A* betűvel indítjuk meg az automatikus ismétlést, akkor a ROM-BIOS *A* betűkből álló sorozatot állít elő, és ezt továbbítja a billentyűzet adatait olvasó program felé. Ha pedig lenyomva tartjuk a váltóbillentyűt (ezt gyakran megtesszük), akkor a ROM-BIOS felismeri a váltó első lenyomását, és a státuszbitet a megfelelő értékre állítja. Az automatikusan ismételt váltóbillentyűjeleket ezután figyelmen kívül hagyja, egészen a váltó felengedésének jeléig. A tanulság mindebből csupán annyi, hogy a ROM-BIOS ésszerű módon kezeli a billentyűzet eseményeit, szükség szerint cselekszik – vagy nyugton marad.

Duplikált billentyűk

Bizonyos billentyűk két példányban is megtalálhatók a billentyűzeten: pl. két csillag van (egyik a felső sorban a 8. billentyűn felül, egy pedig jobbra, a *PrtSc*-billentyűn), két pont, két pluszjel és két mínuszjel, ill. két számjegykészlet, és két (látszatra azonos) *Shift*-billentyű.

A ROM-BIOS igen ésszerűen ugyanazzá a kóddá alakítja át a megduplázott billentyűket. Mindkét csillag a **CHR$(42)**-es karakter lesz. A ROM-BIOS lehetővé teszi a különbség megállapítását is, ha ez számít. A két példányban meglévő karakterek letapogatási kódja megmarad a magasabb helyértékű bájtban. Egy programnak csak ezt kell megvizsgálnia ahhoz, hogy megállapíthassa: melyik billentyűt nyomták le. A két *Shift*-billentyű pedig más bitet állít be a hexa 417-es címen található státuszbájtban. Ha egy program azt akarja kideríteni, hogy melyik *Shift*-billentyűt nyomták le, a megfelelő bitet kell megvizsgálni. (☛ A hexa 417-es cím tartalmát a 3. fejezetben tárgyaltuk.)

Általában az a legjobb, ha egy program nem tesz különbséget a duplikált billentyűk között, bár néhány speciális program különleges célra kihasználja ezt. Ilyen pl. a Microsoft Flight Simulatora és az Ashton Tate Frameworkje.

ASCII-kódok közvetlen bevitele

A PC-billentyűzet és a ROM-BIOS együttesével minden ASCII karakter kódját közvetlen bejuttathatjuk a gépbe. Ha lenyomva tartjuk az *Alt*-billentyűt, akkor a billentyűzet numerikus részén (a jobb oldalon) beírhatjuk a karakter decimális kódját. Ezzel a módszerrel a **CHR$(1)** és a **CHR$(255)** közötti karakterek ASCII kódja vihető be. Az egyetlen ASCII-kód a **CHR$(0)**, nem vihető be ilyen módon, mivel az a nem-ASCII karakterek (pl. a kurzormozgató és a funkcióbillentyűk) jelzésére van fönntartva.

A billentyűzet adatformái

A billentyűzeten lezajlott események átalakítás után egy bájtpárban tárolódnak a ROM-BIOS pufferében. Az alacsony helyiértékű bájtot *fő bájt*nak, a másikat *kiegészítő bájt*nak nevezzük.

Az ASCII-billentyűk

Ha a fő bájt egy **CHR$(1)** és **CHR$(255)** közé eső ASCII karakterértéket tartalmaz, akkor tudhatjuk, hogy vagy az egyik szokásos billentyűzetkaraktert ütötték le, vagy a kibővített ASCII karakterkészletének valamelyik tagját írták be az előbb leírt *Alt-szám* módszer segítségével. (☛ A teljes ASCII-karakterkészlet a *C* függelékben található.) Ezen ASCII-karakterek esetén a kiegészítő bájt a lenyomott billentyű letapogatási kódját tartalmazza. Általában a letapogatási kódnak semmi haszna: A BASIC **INKEY$** függvénye nem adja meg a kiegészítő bájtot. A kiegészítő bájt segítségével lehet megkülönböztetni a duplázott karaktereket. Ha az ASCII-karaktert „mesterségesen" (az *Alt-szám* módszer segítségével) írták le, akkor a kiegészítő bájtban a letapogatási kód értéke 0.

A különleges billentyűk

Ha a fő bájt tartalma nulla (**CHR$(0)**), az azt jelenti, hogy a különleges billentyűvel történt valami. Különleges billentyűnek számítanak a funkcióbillentyűk, a *Shift*tel együtt lenyomott funkcióbillentyűk, a kurzorvezérlő billentyűk (pl. a *Home* vagy az *End*), ill. a *Ctrl-* és az *Alt*-kombinációk közül is több. Ha ezek közül a billentyűk közül egymagában (vagy más billentyűvel együtt) egyet lenyomnak, akkor a kiegészítő bájt a lenyomott billentyűre jellemző egyetlen számértéket tartalmaz. Így saját kódokat definiálhatunk a különleges billentyűkhöz anélkül, hogy összeütközésbe kerülnénk a kibővített ASCII-karakterekkel (ezek **CHR$(128)** és **CHR$(255)** között találhatók). A 6.1. táblázatban látható a 97 különleges karakter értéke, részben logikai, részben pedig szám szerinti sorrendbe állítva.

6.1. táblázat
Az IBM PC billentyűzet 97 különleges billentyűjének kiegészítő bájtértékei. A fő bájt értéke mindig 0

A kiegészítő bájt értéke, dec	A billentyű	A kiegészítő bájt értéke, dec	A billentyű
59	F1	104	Alt-F1
60	F2	105	Alt-F2
61	F3	106	Alt-F3
62	F4	107	Alt-F4
63	F5	108	Alt-F5
64	F6	109	Alt-F6
65	F7	110	Alt-F7
66	F8	111	Alt-F8
67	F9	112	Alt-F9
68	F10	113	Alt-F10
84	Shift-F1	120	Alt-1
85	Shift-F2	121	Alt-2
86	Shift-F3	122	Alt-3
87	Shift-F4	123	Alt-4
88	Shift-F5	124	Alt-5
89	Shift-F6	125	Alt-6
90	Shift-F7	126	Alt-7
91	Shift-F8	127	Alt-8
92	Shift-F9	128	Alt-9
93	Shift-F10	129	Alt-0
		130	Alt-Hyphen
94	Ctrl-F1	131	Alt-=
95	Ctrl-F2		
96	Ctrl-F3	16	Alt-Q
97	Ctrl-F4	17	Alt-W
98	Ctrl-F5	18	Alt-E
99	Ctrl-F6	19	Alt-R
100	Ctrl-F7	20	Alt-T
101	Ctrl-F8	21	Alt-Y
102	Ctrl-F9	22	Alt-U
103	Ctrl-F10	23	Alt-I

A kiegészítő bájt értéke, dec	A billentyű	A kiegészítő bájt értéke, dec	A billentyű
24	Alt-O	15	Fordított Tab
25	Alt-P		(Shift-Tab
30	Alt-A	71	Home
31	Alt-S	72	Nyíl felfelé
32	Alt-D	73	PgUp
33	Alt-F	75	Nyíl balra
34	Alt-G		
35	Alt-H	77	Nyíl jobbra
36	Alt-J	79	End
37	Alt-K	80	Nyíl lefelé
38	Alt-L	81	PgDn
44	Alt-Z	82	Inzert
45	Alt-X	83	Delete
46	Alt-C		
47	Alt-V	114	Echo (Ctrl-PrtSc)
48	Alt-B	115	Ctrl-Nyíl balra
49	Alt-N	116	Ctrl-Nyíl jobbra
50	Alt-M	117	Ctrl-End
3	A tulajdonképpeni nulla	118	Ctrl-PgDn
	karakter,	119	Ctrl-Home
	CHR$(0);		
	l. a szövegben	132	Ctrl-PgUp

MEGJEGYZÉS: Mivel a fő bájtban álló 0 különleges karaktert jelez, a nulla ASCII-karakter (**CHR$(0)**) nem észlelhető. Az IBM definiciója szerint a **CHR$(0)**-t az *Alt – 2* kombinációval kell leírni. Ez a kombináció a 3-as értékű különleges billentyűként jelenik meg: fő bájt = 0, kiegészítő bájt = 3. Ezek szerint elvárható lenne, hogy programjaink a kiegészítő bájt 3-as értékét **CHR$(0)**-ként értelmezzék. Tapasztalataink szerint viszont egyetlen programozási nyelv vagy program sem veszi a fáradságot, hogy így járjon el.

A teljes karakterkészlet és a különleges billentyűk kódjait a ROM-BIOS generálja, de a programozási nyelvek különböző módon használják fel a kódokat. A BASIC pl. nem egységesen kezeli a különleges billentyűket. Ha közönséges utasításokkal olvassuk be az adatokat, akkor a BASIC a szabályos ASCII-karaktereket átadja a BIOS-nak, a különleges billentyűket pedig kiszűri. Néhány különleges billentyűre az **ON KEY** utasítással reagálhatunk, de az **INKEY$** utasítással közvetlenül a ROM-BIOS-tól is megtudhatjuk a billentyűk kódját. Az utóbbi módszerrel kideríthető, milyen különleges billentyűt nyomtak le. Ha az **INKEY$** egybájtos eredményt ad, akkor vagy a közönséges, vagy a kibővített ASCII-karakterkészletből egy karaktert tartalmaz. Ha az **INKEY$** eredménye 2 bájt hosszú, akkor az 1. bájt a ROM-BIOS fő bájtja, értéke **CHR$(0)**; a 2. bájt a kiegészítő bájt, ez tartalmazza, hogy melyik különleges billentyűt nyomták le.

A billentyűzet vezérlése

A billentyűzet működésének és a billentyűzetről jövő adatok gyűjtésének a ROM-BIOS-ban lévő programjai adatterületeket is használnak. Ezek a tár alacsony címein (hexa 417–472.), valamint a hexa 412. és hexa 488. bájtokban találhatók. Programjaink ezeket a területeket felhasználhatják a billentyűzet állapotának megvizsgálására vagy működésének módosítására. Most sorra vesszük azokat a címeket, amelyeknek a tartalmát jól fel lehet használni, és azokat, amelyeket biztonsággal megváltoztathatunk.

A státuszbájtok

Kezdjük a két státuszbájttal. Ezek a hexa 417. (6.2. táblázat) és a 418. (6.3. táblázat) címen találhatók. A két bájtban külön biteken van kódolva, hogy melyik váltó- vagy záróbillentyű aktív. A PC-család minden szabványos modelljén megtaláljuk ezt a két bájtot. [JR] jelenleg az egyetlen eltérés a 2. bájt 2. bitjén van. Ez az óraütés nevű bit a PCjr sajátossága. Más szempontból a bitformátum egységes az összes szabványos PC-modellen.

6.2. táblázat
A hexa 417. címen található 1. billentyűzet státuszbájt kódolása

Bit 7 6 5 4 3 2 1 0	Jelentése
X	Inzertbillentyű: 1: aktív; 0: nem aktív
. X	Nagybetűzár *(Caps Lock)*: 1: aktív; 0: nem aktív
. . X	Számzár *(Num Lock)*: 1: aktív; 0: nem aktív
. . . X	Képernyőzár *(Scroll Lock)*: 1: aktív; 0: nem aktív
. . . . X . . .	*Alt*-váltó 1: aktív (*Alt* lenyomva); 0: nem aktív
. X . .	*Crtl*-váltó: 1: aktív (*Ctrl* lenyomva); 0: nem aktív
. X .	Közönséges váltó: 1: aktív (bal oldali *Shift*-billentyű lenyomva); 0: nem aktív
. X	Közönséges váltó; 1: aktív (jobb oldali *Shift*-billentyű lenyomva); 0: nem aktív

6.3. táblázat
A hexa 418. címen található 2. billentyűzet státuszbájt kódolása

Bit 7 6 5 4 3 2 1 0	Jelentése
X	1: *Ins* lenyomva
. X	1: *Caps Lock* lenyomva
. . X	1: *Num Lock* lenyomva
. . . X	1: *Scroll Lock* lenyomva
. . . . X . . .	1: Felfüggesztési állapot aktív *(Ctrl-Num Lock)*
. X . .	1: PCjr-nél a billentyűzet hangot ad
. 0 .	Nincs felhasználva
. 0	Nincs felhasználva

A beszúrási (insert) állapot

A ROM-BIOS az 1. bájt 7. bitjén tartja nyilván ezt az állapotot. A szerző által ismert programok kivétel nélkül figyelmen kívül hagyják ezt a bitet, és maguk tartják nyilván, hogy a beszúrás ki van-e kapcsolva. Ezért – bár lehetséges – nem szokás a ROM-BIOS-ban levő beszúrási állapotot jelző bitet használni.

A nagybetűzár (Caps-Lock) állapota

Sok programozó erőszakosan aktivizálja a nagybetűzár állapotát az 1. bájt 6. bitjének beállításával. Ez megzavarhatja a felhasználókat, de ennek ellenére megbízhatóan működik, és sokan használják is ezt a fogást.

A billentyűzet felfüggesztés (Hold) állapota

A n billentyűzetnek ez az állapota a PC érdekes tulajdonsága, bár nincs gyakorlati kapcsolata programjainkkal. A BIOS többek között a *Ctrl-Num Lock* billentyűkombinációt is figyeli. Ha a BIOS ezt a kombinációt észleli, akkor az 1. bájt 3. bitjének beállításával átmegy az ún. billentyűzetfelfüggesztési állapotba. Ebben az állapotban a BIOS-program egy nyomtatható karakter lenyomásáig vár; a gép vezérlését addig nem adja vissza a futó programnak, míg nem kap nyomtatható karaktert. Ezt a sajátosságot a gép működésének felfüggesztésére használhatjuk fel.

A billentyűzet felfüggesztése alatt a megszakításokat a számítógép szokásosan kezeli. Ha pl. a lemezmeghajtó megszakítást generál (egy lemezes művelet befejezését jelzi), akkor a lemezes megszakítások kezelője megkapja és feldolgozza a megszakítási jelet. Amikor azonban a megszakítás kezelője befejezi a működését, akkor annak a programnak adja vissza a vezérlést, amelyik a megszakítás megjelenésének pillanatában futott (azaz a BIOS billentyűzetkezelő részében futó végtelen üres ciklusnak). Tehát a billentyűzet felfüggesztése alatt a számítógép képes reagálni a külső megszakításokra de a programok általában teljesen fel vannak függesztve. A BIOS billentyűzetet irányító része továbbra is kezeli a billentyűzetről származó megszakításokat, és amint egy közönséges billentyű lenyomását észleli, pl. a szóközét vagy egy funkcióbillentyűét – a váltóbillentyű lenyomása azonban egyedül nem elég –, befejezi a billentyűzet felfüggesztését. Ezután a vezérlést annak a programnak adja vissza, amelyik eredetileg is futott.

A billentyűzet felfüggesztésének a programozás szempontjából nincs gyakorlati jelentősége, de a felhasználó számára általános módszerként használható a programfutás felfüggesztése.

Tudnunk kell, hogy a billentyűzet felfüggesztése nem „bombabiztos" fogás. Előfordulhat, hogy egy program külső megszakítás (pl. az óraütések) hatására ebben az állapotban is fut. Ha egy program meg akarja akadályozni, hogy felfüggesszék, akkor létrehozhat egy olyan megszakításkezelőt, amelyik átsegíti a felfüggesztett állapoton; vagy egyszerűen kikapcsolja azt, amint a felfüggesztés bekapcsolását érzékeli.

A záróbillentyűk állapota

A 4–7. bitek mindkét bájtban ugyanazokat a billentyűket jelentik. Az 1. bájtban a záróbillentyűk pillanatnyi állapotát mutatják, a 2. bájtban azt, hogy a megfelelő billentyű most éppen le van-e nyomva.

Ezeket a biteket tetszés szerint kiolvashatjuk, de kérdéses, hogy a programban használhatók-e valamire. Némileg eltekintve a *Caps Lock* billentyűtől, nem bölcs dolog a váltóbillentyűk állapotát megváltoztatni. Ezek az 1. bájt 4–6. bitjén találhatók. Az pedig egyenesen káros lehet, ha a billentyűk lenyomott állapotát jelző biteket változtatjuk meg. (Az 1. bájt 0–3. bitjéről és a 2. bájt 4–7. bitjéről van szó.)

Megjegyzések

Aki részletesebben akarja ismerni a PC billentyűzetének működését, az tanulmányozza az IBM Technical Reference kézikönyvben a ROM-BIOS listáját, de vigyázzon, nehogy elkövessen egy tipikus hibát. Ezt ugyanis a ROM-BIOS (azon belül is elsősorban a ROM-BIOS által használt) megszakítások tanulmányozása közben gyakran elkövetik. A ROM-BIOS két megszakítást használ a billentyűzethez: az egyik a billentyűzet megszakításaira válaszol (a 9-es), és a billentyűzet adatait gyűjti a tár alacsony című részén levő pufferbe. A másik a billentyűzet szolgáltatásait kérelmező megszakításra válaszol (a 22-es), és ez adatokat ad át a pufferből a programoknak és a DOS-nak. A két megszakítást könnyű összekeverni, és még könnyebb ezt tovább keverni a *Break*-billentyű megszakításaival (a 27-es és a 35-ös). A 6.4. táblázat összefoglalja a billentyűzet megszakításait.

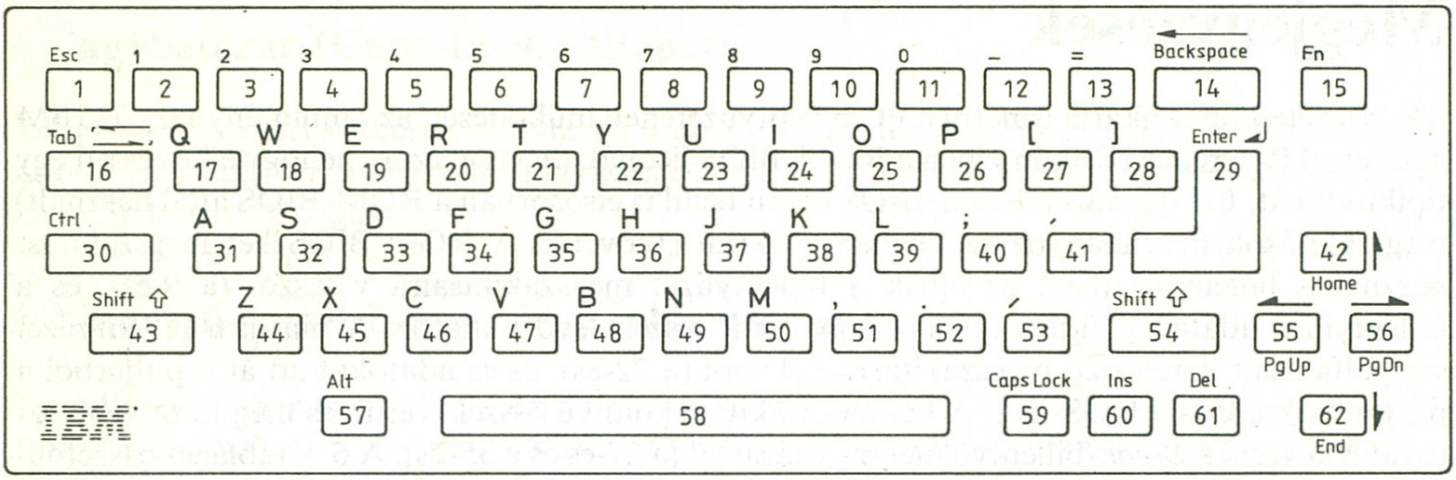
A könyvben gyakran felbukkan az a jótanács, hogy ne kockáztassunk: játsszunk szabályosan. Inkább az egész PC-családban használható programokat írjunk, nem pedig az egyes modellek rigolyáihoz igazodókat; a hardver közvetlen programozása helyett használjuk a legális eszközöket az adatok kezelésére (pl. a ROM-BIOS nyújtotta szolgáltatásokat). Ezek a szabályok a billentyűzet programozására éppúgy érvényesek, mint bármilyen más típusú programozásra.

6.4. táblázat
A billentyűzethez kapcsolódó megszakítások

Megszakítás		Megszakítás forrása	Jelentése
Decimális	Hexadecimális		
9	9	Billentyűzet	Jelzi, hogy valami történt a billentyűzeten
22	16	ROM-BIOS	Behívja a szabványos BIOS billentyűzetszolgáltatásait (l. 12. fejezet)
27	18	ROM-BIOS	Megszakítást generál, ha megszakításbillentyű kombinációt nyomnak meg a BIOS vezérlése alatt; ha megírjuk, egy szubrutin hívása megy végbe
35	23	DOS	Ha megírjuk, akkor meghívódik egy szubrutin abban az esetben, ha a DOS vezérlése alatt egy megszakításbillentyű-kombinációt nyomnak le

A PCjr sajátosságai

JR A PCjr-t úgy tervezték, hogy a lehető legpontosabban utánozza az eredeti 83 billentyűs PC-t. A 6.2. ábrán azonban látható, hogy a PCjr saját billentyűzetén csak 62 billentyű van, tehát nem egyezik meg teljesen a PC billentyűzetével. A feladatot néhány ravasz billentyűcsellel sikerült megoldani.



6.2. ábra. Az IBM PCjr billentyűzete és a letapogatási kódok

A PCjr 61 billentyűje azonos a PC-vel, és van egy új billentyű is: az *Fn* (funkció). A hiányzó 22 billentyűt a PCjr billentyűinek valamilyen kombinációjával állítjuk elő. Ez korántsem olyan egyszerű, mint első látásra gondolnánk, mert a PC billentyűs műveleteinek PCjr megfelelői nem igazán kézenfekvőek. (L. a 6.5. táblázatot.)

A PCjr-nél találunk 5 olyan sajátos billentyűkombinációt, amelyiknek a PC-nél nincs megfelelője. Ezeket soroljuk fel a 6.6. táblázatban. Az ötödiket (a *Shift-Fn-Esc*-t) a PCjr dokumentációja épphogy megemlíti.

6.5. táblázat
A 22 hiányzó PC billentyű megfelelői a PCjr-en

PC	PCjr
F1...F10	*Fn*-billentyű, majd 1...0
	Alt-[
	Alt-/
	Alt-'
PrtSc	*Fn-P*
* (A PrtSc billentyűn)	*Alt-*. (Nem azonos *Shift-8* csillaggal)
Ctrl-PrtSc (echó)	*Fn-E*
Num Lock	*Alt-Fn-N*
Ctrl-Num Lock (szünet)	*Fn-Q*
Scroll Lock	*Fn-S*
Break	*Fn-B*

PC	PCjr
Home	*Fn*-nyíl felfelé
Ctrl-Home	*Ctrl-Fn*-nyíl felfelé
PgUp	*Fn*-nyíl balra
Ctrl-PgUp	*Ctrl-Fn*-nyíl balra
5 (a numerikus billentyűkön)	Nincs megfelelője
End	*Fn*-nyíl lefelé
Ctrl-End	*Ctrl-Fn*-nyíl lefelé
PgDn	*Fn*-nyíl jobbra
Ctrl-PgDn	*Ctrl-Fn*-nyíl jobbra
– (a numerikus billentyűkön)	*Fn*-kötőjel
+ (a numerikus billentyűkön)	*Fn*=

6.6. táblázat
A PCjr sajátságos billentyűkombinációi

Billentyűkombináció	Jelentése
Ctrl-Alt-nyíl balra	Balra lépteti a képernyőt
Ctrl-Alt-nyíl jobbra	Jobbra lépteti a képernyőt
Ctrl-Alt-Caps Lock	Ki-, ill. bekapcsolja a billentyűzet hangját
Ctrl-Alt-Ins	Diagnosztikai programokat hív be
Shift-Fn-Esc	A számjegybillentyűk funkcióbillentyűkként viselkednek

A PCjr billentyű műveletei

A PCjr billentyűzetének működési fázisai többé-kevésbé azonosak a PC-nél megismertekkel. (☛ L. a billentyűzet működését.) Mivel azonban a PCjr billentyűzete más, bevezettek egy teljesen új – de ismerős – réteget a műveletek tetejére.

A PCjr billentyűzetek eseményei – akárcsak a PC-nél – megszakítást okoznak. Ez azonban különleges megszakítás: a 2-es nemmaszkolható megszakítás (NMI). Ez viszont a hexa 48-as megszakítást hívja, amelyik a 62 billentyűs letapogatási kódot a megfelelő 83 billentyűssé alakítja át. A hexa 48-as megszakítás generál még egy 9-es megszakítást is (a PC-billentyűzet megszakítását utánozva), és innentől minden a lehető legpontosabban követi a PC szabványát; azaz a ROM-BIOS lefordítja a PC eseménykódjait a végső értelmezésnek megfelelőre.

Erre az előfeldolgozásra azért van szükség, mert a PCjr billentyűzetét nem segíti 8048-as mikroprocesszor. A PC billentyűzete elég okos ahhoz, hogy maga is tárolni tudja az események kódját. Ez teszi lehetővé a billentyűzet megszakításának letiltását, amikor a számítógép más részei igénylik a figyelmet. A PCjr nem rendelkezik ilyen képességekkel, ezért a billentyűzetén végbemenő események sürgetőbbek, és minden más megszakítást megelőznek.

A PCjr és nagyobb teljesítményű testvérei között az egyik legnagyobb különbség, hogy a ROM-BIOS működteti a billentyűzet és a lemezmeghajtók kiszolgálását. Ebből konfliktus származik, mert a ROM-BIOS a lemezt részesíti előnyben, mivel ez nem rendelkezik közvetlen memóriahozzáféréssel. Amikor a lemez működik, a megszakítás a teljes rendszerben le van tiltva, ezért a billentyűről az adatbevitel is szünetel. (A soros porton át is szünetel az aszinkron adatátvitel.)

A PCjr-billentyű státuszbájtja

A PCjr-nek az előzőekben említett kettőn kívül van egy harmadik státuszbájtja is. Ez a hexa 488-as címen található. Ennek a bájtnak a PCjr 62 billentyűs billentyűzete szempontjából van különös jelentősége.

A státuszbájt egyes bitjeinek jelentése a 6.7. táblázatban látható. Általában semmi előny nem származik e bitek kiolvasásából vagy megváltoztatásából, azt azonban jó tudni, hogy az automatikus billentyűismétlés letiltható a 3. bit 1-re állításával. Ezenkívül a 2. bit 1-re állításával megduplázhatjuk az ismétlések között és az ismétlés megkezdéséig eltelt időt. Ha valaki a

6.7. táblázat
A hexa 488. címen található PCjr billentyűzet státuszbájt kódolása

Bit 7 6 5 4 3 2 1 0	Jelentéstábla
X	Funckciójelző
. X	Megszakításbillentyűt jelez *(Fn-B)*
. . X	Függőben levő funkció: *Fn* lenyomva
. . . X	Funkciózár: a numerikus billentyűk funkcióbillentyűként viselkednek
. . . . X . . .	Az automatikus ismétlést szabályozza: 0: engedélyezve; 1: tiltva
. X . .	Az automatikus ismétlés sebességét szabályozza 0: fél sebesség; 1: teljes sebesség
. X .	Az automatikus ismétlés megkezdése előtti időt szabályozza 0: engedélyezve; 1: tiltva
. X	Jelzi, hogy az ismétlőbillentyű generálása aktuális

PCjr-en kísérletezni szeretne az automatikus ismétlés elnyomásával, az tegye be a DOS kisegítő programjainak lemezét, majd írja le a következő parancsokat:

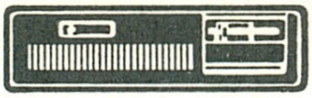
```
DEBUG
F 0:488 L 1 08
```

Ezek a parancsok csak a PCjr-nél hatásosak.

Javaslatok a PCjr programozásához

A PCjr billentyűzetét úgy tervezték, hogy az a PC teljes billentyűzetét emulálja, és azt más billentyűzetszerű berendezéssel váltsák vagy javítsák föl. Ez lehet pl. egy egér vagy egy teljes billentyűzet. Ennek a tervezési elképzelésnek a fényében különösen rövidlátók vagyunk, ha programjainkat a PCjr billentyűzetének sajátosságai szerint tervezzük. Egyetlen esetben vehetjük mégis figyelembe a PCjr billentyűzetének sajátosságait: a programban használt különleges billentyűk kiválasztásakor. Sok olyan programban, amelyet az eredeti PC-re terveztek, a billentyűk használata – különösen a funkcióbillentyűké – a PC teljes billentyűzetére illeszkedik. A PCjr megjelenése után ajánlatos a billentyűk kiválasztását újra átgondolni, a két billentyűzet annyira eltérő. Vagy a PCjr, vagy a PC szempontjából legjobb megoldást választjuk, esetleg a kettő közötti kompromisszumot, sőt a program működését ahhoz a géphez is igazíthatjuk, amin éppen dolgozunk. (☛ L. a gép azonosítójának megállapítását.) A választás a céltól és a program lehetséges alkalmazásaitól függ.

Az AT sajátosságai

AT 

Az AT billentyűzete is eltér a szabványos PC-jétől. A PCjr billentyűzetétől eltérően a felhasználó számára látható és a programokkal érzékelhető különbségek itt igen kicsik. Ami azonban a hardvert és a BIOS-t illeti, az AT billentyűzete a PCjr billentyűzetéhez hasonlóan szimulálja a PC billentyűzetét. Az AT billentyűzetének belsejében található különbségek programjaink számára lényegében láthatatlanok, nem is foglalkozunk velük részletesen.

Az AT billentyűzetének elrendezése, néhány billentyű áthelyezésétől és egy új billentyűtől eltekintve, azonos az eredeti PC billentyűzetének elrendezésével. A billentyűk áthelyezése miatt általában nem kell megváltoztatni a billentyűzet használatának programozását vagy az irányításra kiválasztott billentyűket. Jegyezzük meg, hogy a program irányításánál gyakran használt *Esc*-billentyű teljesen új helyre került: a bal fölső sarokból a jobb fölsőbe. Ez igen zavaró annak, aki a PC és az AT billentyűzetét egyaránt használja.

Az AT billentyűzetén új a *Sys Req*-billentyű. Ezt a billentyűt nem a programozók használják. Az a feladata, hogy a rendszer a programtaskok között váltson, ha az AT többtaskos üzemmódban működik, az AT 286-os processzorának különleges lehetőségét is kihasználva. Ez a billentyű tabu a PC-s programok számára.

Az AT és a billentyűzete között kétirányú hardverkapcsolat van. A billentyűzet információkat küld az AT felé, az AT pedig parancsokat küld a billentyűzetnek (pl. gyújtsa ki a jelzőfényeket). Értelmetlen lenne ezekkel a billentyűzetvezérlő parancsokhoz hozzányúlni.

Érdemes megjegyezni, hogy a francia, a spanyol, az olasz és a brit billentyűzetváltozatokat az AT-vel egyidőben mutatták be.

Kockázatos dolog megpróbálkozni az IBM lépéseinek kitalálásával, mert az AT-billentyűzet elrendezése – a nemzeti változatokkal együtt – az IBM PC-családjának jövendő termékeinél is szabvány marad. Így lesz-e vagy sem, igazából nem lényeges a számunkra, hiszen programozási szempontból az AT billentyűzete lényegesen nem különbözik a PC billentyűzetétől; a PCjr billentyűzeténél viszont valóban van néhány lényeges eltérés.

7 A hanggenerálás

Bevezetés • 126

A hang fizikája • 126

Hanggenerálás a számítógéppel • 127

Hanggenerálás az időzítő IC-vel • 127
A hangszóró közvetlen vezérlése • 130

A hangerő és a hang minősége • 130

Hanggenerálás a PCjr-rel • 131

A TI hang-IC • 132
A hang-IC vezérlése • 134
Hangkimenet a PCjr-en • 136

Bevezetés

A PC-család minden tagja alkalmas egyszerűbb hangok előállítására a számítógép programozható időzítő IC-je (a 8253-5) és a beépített hangszóró segítségével. [JR] A PCjr-nek bővebb hangadási lehetőségei vannak: egy különleges hangkeltő IC, további hangforrások és kiegészítő hangkimenetek. Mivel ez utóbbiak a PCjr sajátosságai, ezekkel röviden külön is foglalkozunk a fejezet végén. A fejezet első részében a hanggenerálásnak azokkal a sajátosságaival foglalkozunk, amelyek a PC-család egészére jellemzők.

A számítógépes hangelőállítás megértése miatt először a hang néhány alapvető tulajdonságát kell ismerni. Ezzel foglalkozunk először.

A hang fizikája

A hang nem más, mint a légnyomás szabályos pulzálása vagy vibrálása. Hang akkor keletkezik, ha egy rezgőforrás rezgésbe hozza a levegő részecskéit. A rezgőforrás mozgása láncszerű jelenséget hoz létre, amely a forrástól távolabbra is eljuttatja a levegő rezgését. Ezt a mozgást hanghullámnak nevezzük.

7.1. táblázat
Négy oktávnyi zenei hang frekvenciái

Hang	Frekvencia	Hang	Frekvencia	Hang	Frekvencia	Hang	Frekvencia
C0	16,35	C2	65,41	C4	261,63	C6	1046,50
C#0	17,32	C#2	69,30	C#4	277,18	C#6	1108,73
D0	18,35	D2	73,42	D4	293,66	D6	1174,66
D#0	19,45	D#2	77,78	D#4	311,13	D#6	1244,51
E0	20,60	E2	82,41	E4	329,63	E6	1328,51
F0	21,83	F2	87,31	F4	349,23	F6	1396,91
F#0	23,12	F#2	92,50	F#4	369,99	F#6	1479,98
G0	24,50	G2	98,00	G4	392,00	G6	1567,98
G#0	25,96	G#2	103,83	G#4	415,30	G#6	1661,22
A0	27,50	A2	110,00	A4	440,00	A6	1760,00
A#0	29,14	A#2	116,54	A#4	466,16	A#6	1864,66
B0	30,87	B2	123,47	B4	493,88	B6	1975,53
C1	32,70	C3	130,81	C5	523,25	C7	2093,00
C#1	34,65	C#3	138,59	C#5	554,37	C#7	2217,46
D1	36,71	D3	146,83	D5	587,33	D7	2349,32
D#1	38,89	D#3	155,56	D#5	622,25	D#7	2489,02
E1	41,20	E3	164,81	E5	659,26	E7	2637,02
F1	43,65	F3	174,61	F5	698,46	F7	2793,83
F#1	46,25	F#3	185,00	F#5	739,99	F#7	2959,96
G1	49,00	G3	196,00	G5	783,99	G7	3135,96
G#1	51,91	G#3	207,65	G#5	830,61	G#7	3322,44
A1	55,00	A3	220,00	A5	880,00	A7	3520,00
A#1	58,27	A#3	233,08	A#5	932,33	A#7	3729,31
B1	61,74	B3	246,94	B5	987,77	B7	3951,07
						C8	4186,01

Megjegyzés: Egyenletesen temperált kromatikus skála: A4 = 440.
Szabványos amerikai skála: Az Amerikai Szabványügyi Hivatal 1936-ban fogadta el.

Az IBM PC-k beépített hangszóróját a számítógép által előállított elektromos impulzusok hozzák rezgésbe. Mivel a számítógép rendszerint bináris számokkal dolgozik, az általa előállított feszültségimpulzusok is kétállapotúak. A feszültségszintek közötti átmenetek mozgatják a hangszóró membránját. Hang akkor keletkezik, amikor a hangszóróba jutó feszültség előbb alacsony szintről magasra, majd ismét alacsonyra vált; ennek hatására a membrán előbb kifelé, majd befelé mozog. Ezt az egyetlen rezgést – ami egy kifelé, majd egy befelé irányuló impulzusból áll – ciklusnak vagy periódusnak nevezzük, és hertzben mérjük. (Egy hertz másodpercenként egy periódus.) A PC hangszórójából egyetlen hangperiódust egy kattanásnak hallunk. Folytonos hangot akkor észlelünk, ha másodpercenként több periódust küldünk a hangszóróra. Ha növeljük a ciklusok számát, akkor az impulzusok összemosódnak, és egy adott tónusú hangot hallunk. Ha pl. a membránt másodpercenként 523-szor mozgatjuk ki és be (523 hertzes ütemben) akkor a kétvonalas C-ként ismert zenei hangot halljuk.

Az átlagember 20 hertz és 2000 hertz közötti hangot hall. Az IBM PC hangszórójával 13 hertz és több, mint egymillió hertz közötti hangokat lehet előállítani: ez messze felülmúlja az emberi hallás felső határát. Ezt a tartományt akkor tudjuk igazán értékelni, ha az átlagos emberi hanggal hasonlítjuk össze: ez a 125 és az 1000 hertz közötti tartományba esik.

Az IBM személyi számítógép hangszórójának hangereje nem szabályozható, ezért nem is feladata a tökéletes hang előállítása. Ennek következtében a különböző frekvenciák eltérő hatást keltenek: néhányat másoknál hangosabbnak hallunk; bizonyos hangmagasságokat egészen tisztán meghallunk, egyes hangokat viszont egyáltalán nem érzékelünk. Ez az ingadozás a hangszóró tervezéséből ered, nem tudjuk befolyásolni.

Hanggenerálás a számítógéppel

A hangszóróval kétféle módon adhatunk hangot – a hangforrások közül az egyiket, a másikat vagy mindkettőt használva. Az egyik módszernél olyan programot kell írni, amelyik a programozható perifériaillesztő IC-ben (PPI) található hangszóróvezérlő bitek segítségével ki-, ill. bekapcsolja a hangszórót. Ha ezt a módszert (a programmódszert) használjuk, akkor programunk az impulzus időzítését – tehát frekvenciáját – szabályozza. A másik módszer (az időzítős módszer) a PC beépített programozható időzítőjét – a 8253-5 jelű IC-t – használja fel a membrán egy meghatározott frekvenciával való rezegtetésére. Az időzítő IC alkalmazása mellett két érv is szól: egyrészt így a hangszórót az időzítő IC vezérli, és nem egy program – a CPU tehát a számítógép más feladataival is foglalkozhat –, másrészt az időzítő IC független a CPU sebességétől – az AT-nél nagyobb, a PCjr-nél viszont kisebb.

A programmódszer és az időzítős módszer váltogatásával vagy együttes felhasználásával az egyszerű és az összetett hangzások széles skálája állítható elő. A következőkben részletesen ismertetjük az időzítő IC és a hangszóró közvetlen programozásával létrejövő hanggenerálást. Végül leírjuk azokat az új lehetőségeket, amelyekkel a PCjr bővítette a PC-családot.

Hanggenerálás az időzítő IC-vel

A 8253-5 típusú programozható időzítő a szabványos PC-k hanggenerálásának, de egyben a rendszerórának is a lelke. Elsősorban hanggenerátorként foglalkozunk vele, bár a 8253–5 áramkört nem véletlenül nevezik időzítőnek. Elsődleges feladata ugyanis az idő nyilvántartása, éppúgy az időt jelzi, mint ahogy a metronom a zenészeknek.

Az áramkör működése: A 8253 a rendszer órájától – a 8284A-tól – kap egy jelet. Ez utóbbi másodpercenként 1 193 180-at rezeg, azaz frekvenciája 1193 megahertz (MHz). Az időzítő úgy van programozva, hogy a vezérlőóra minden 65 536. periódusában egy megszakítást generáljon. Ez másodpercenként átlagosan 18,2 megszakítást jelent. Ezt az óramegszakítást óraütésnek szokták nevezni. A ROM-BIOS tartja nyilván az óraütéseket, ebből számítja ki a napi időt, minden óraütéskor eggyel növeli saját számlálóját. Ezenkívül saját (ún. órajel) megszakítását is generálja (28-as megszakítás).

Sok program a ROM-BIOS órajel-megszakítását használja az idő nyilvántartására, de vannak olyan programok is, amelyek ezt megkerülik, és közvetlenül az időzítő IC-t hsználják. A BASIC pl. közvetlenül az időzítő IC segítségével méri a hangok óraütés egységben megadott hosszát. Mivel azonban a másodpercenkénti 18,2 óraütés gyakran nem elég szapora az egyes zeneműveknél megkövetelt pontosság eléréséhez, a BASIC átprogramozza az órát úgy, hogy négyszer gyorsabban járjon. Így a 8-as megszakítás másodpercenként 72,8-szor fordul elő a szokásos 18,2 helyett. Amikor a BASIC ezzel a négyszeres ütemmel számol, pontosabban képes visszaadni egy zenedarab előírt tempóját.

🕮 MEGJEGYZÉS: a BASIC a **MUSIC**-utasítás időtartamára megnégyszerezi az óra frekvenciáját. A BIOS órajel-megszakításának megzavarását úgy kerüli el, hogy a 8-as megszakítás vektorát egy olyan szubrutinra állítja át, amelyik minden negyedik óraütésnél jelez a ROM-BIOS-nak. A negyedik ütéssel a megszakítás kezelője átmenetileg átadja a vezérlést a BIOS-nak, hogy az előírásszerűen megnövelhesse a számlálóját, és kiadhassa a 28-as megszakítást; ezután a vezérlést a BASIC-nek adja vissza.

Az időzítő IC programozása

Az időzítő IC-vel a hangot két lépésben lehet létrehozni: először be kell programozni az időzítőt a megfelelő frekvenciára, majd az időzítő kimenetét a hangszóróra kell átirányítani. A két lépést külön-külön is elvégezhetjük. A hang kibocsátása akkor kezdődik, amikor mindkét lépést végrehajtottuk, és akkor áll le, ha bármelyiket megszüntetjük.

Az időzítőt tetszőleges frekvenciájú impulzusok kiadására programozhatjuk be, mivel azonban nem tartja nyilván, hogy a hang meddig szól, mindaddig szólni fog, amíg ki nem kapcsoljuk. Ezért programunknak kell gondoskodni arról, hogy megfelelő idő múltán a hangot kikapcsolja. Az eltelt időt egy időzítőutasítással mérhetjük.

Az időzítő hangját éppúgy a megfelelő szám megadásával programozzuk, mint a BASIC-ben az óraütések generálását. A parancs hatására az időzítő számolni kezdi a rendszeróra impulzusait – a rendszeróra frekvenciája 1193 MHz –, míg az általunk megadott számút le nem számlálta. Ekkor megszakítás helyett egy impulzust ad ki, és újra nulláról kezdi a számlálást. Valójában az időzítő az óra jelét osztja a mi számunkkal, így hozza létre a kimenő frekvenciát. Amint a hangszórót bekapcsoljuk, az időzítő egy impulzussorozatot bocsát ki, az pedig a megadott frekvenciájú hangot ad.

A számlálóérték és az eredőfrekvencia között az alábbi kétirányú kapcsolat áll fenn:

Számláló = 1 193 180 ÷ Frekvencia
Frekvencia = 1 193 180 ÷ Számláló

A képletekből látható, hogy alacsony frekvenciájú (mély) hanghoz magas számlálóérték, nagyfrekvenciájú (magas) hanghoz pedig kis számlálóérték tartozik. A 100-as számlálóérték kb. 11 931 Hz-es magas hangot, a10 000 számlálóérték 119 Hz-es mély hangot eredményez.

A 16 bites aritmetikán belül szinte tetszőleges frekvenciát elő tudunk állítani. A legalacsonyabb frekvencia 18,2 Hz – ehhez a 65 535 számláló tartozik; a legmagasabb pedig 1193 MHz – ekkor a számlálóérték 1. A BASIC ezt a gyakorlati szempontból kielégítő 37 – 32 767 Hz-es határok közé korlátozza. A következő program azt mutatja be, hogy a beépített hangszóró frekvenciaátfogása még annál is kisebb, mint amit a BASIC biztosít.

Ha kiszámítottuk a kívánt frekvencia előállításához szükséges számlálóértéket, azt kiküldjük a 8253-as időzítő regisztereibe. Ezt három különböző porthoz fordulással oldjuk meg. Az első azt tudatja az időzítővel, hogy számlálóérték következik: ezért a 67-es portra a 182 értéket (hexa B6) küldjük. A következő két kimeneti művelettel a számláló alacsony, majd magas helyiértékű bájtját küldjük ki. Ez egy 16 bites előjeltelen szám, amit a 66-os portra küldünk ki. A folyamat BASIC-ben:

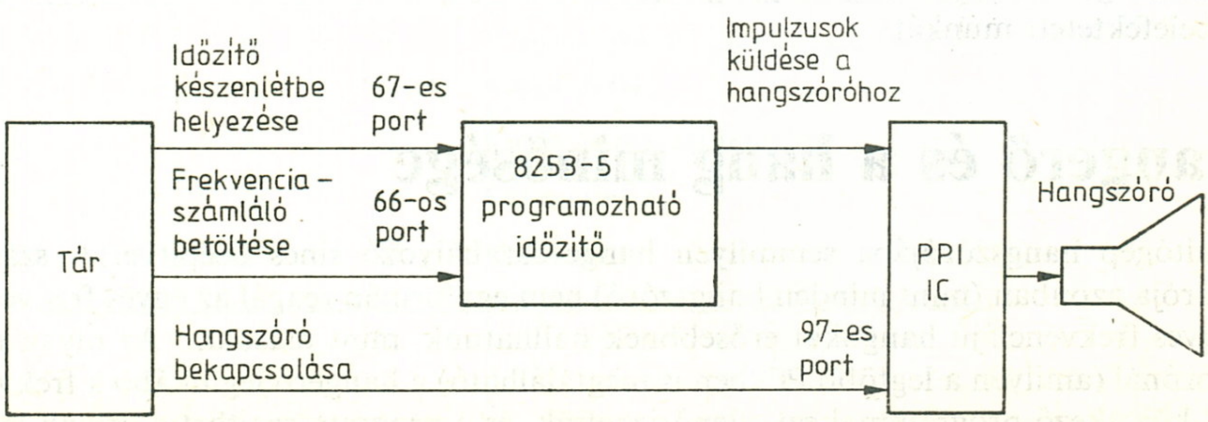
```
10 SZAML=1193280!/3000          '3000 Hz a frekvencia
20 AL.SZAML=SZAML MOD 256       'alacsony helyiértékű bájt
30 MA.SZAML=SZAML \ 256         'magas helyiértékű bájt
40 OUT 67, 182                  'felkészíti az időzítőt
50 OUT 66, AL.SZAML             'alacsony bájt betöltése
60 OUT 66, MA.SZAML             'magas bájt betöltése
```

A hangszóró bekapcsolása

Az időzítő beprogramozása után még be kell kapcsolni a hangszórót, hogy az időzítő által előállított jelet hasznosíthassuk. Akárcsak a PC legtöbb más részénél, a hangszórónál is bizonyos értéket kell megadott portokra küldeni. Ezt a folyamatot szemlélteti a 7.1. ábra. A hangszóró a programozható perifériaillesztő (PPI) felügyelete alatt áll, és a 97-es (hexa 61) portot használja. A port 8 bitjéből a hangszóró csak kettőt használ: a 0. és az 1. alacsony helyiértékű biteket. A többi bit más célra való, tehát fontos, hogy ne rontsuk el őket, ha a hangszóróval foglalkozunk.

A legkisebb helyiértékű bit, a 0., az időzítőnek azt a jelét vezérli, amelyik a hangszórót hajtja meg. A másik bit, az 1., a hangszóró pulzálását vezérli. Mindkét bitre az 1 értéket kell írni, hogy a hangszóró engedelmeskedjen az időzítőnek. A többi bit elrontása nélkül a következő módon állíthatjuk a biteket 1-re:

```
70 REGI.PORT=INP(97)               '97-es port előző értéke
80 UJ.PORT=(REGI.PORT OR &H03)     'a bitek 1-re állítása
90 OUT 97, UJ.PORT                 'a hangszóró bekapcsolása
```



7.1. ábra. A 8253-5 időzítő és a hangszóró a hangfrekvencia előállítása

A hangszóró közvetlen vezérlése

Az időzítő úgy vezérli a hangszórót, hogy egy impulzussorozattal a hangszórót pulzáló mozgásra készteti. Ugyanezt a hatást egy olyan programmal is elérhetjük, amelyik követlenül a hangszóróhoz küld ki- és bekapcsoló jeleket. Pontosabban azt tesszük, hogy a 97-es port (hexa 61) 0. bitjét nullára állítjuk (ezzel a hangszórót bekapcsoljuk), majd váltakozva 0-t és 1-et küldünk az 1. bitekre, ez pulzáló mozgásba hozza a hangszórót. Ha ezt a módszert használjuk, akkor a program időzítése szabja meg a frekvenciát. Minél gyorsabban fut a program, annál magasabb a hang. A következő BASIC-program szemlélteti a módszert. (A példában feltételezzük, hogy a 97-es port (hexa 61) értéke 76):

```
10 X=INP(97) AND &HFC          'a port értékének utolsó 2 bitje 0
20 OUT 97,X                    'a membrán behúzása
30 OUT 97,X+2                  'a membrán kilökése
40 GOTO 20
```

A 20-as és a 30-as sorban található utasítások befelé, ill. kifelé mozgatják a membránt. Mindegyik egy fél periódust jelent, a kettő együttesen egy teljes hangciklust.

A példa olyan gyorsan fut, ahogy a BASIC megengedi, és így a lehető legmagasabb hangot állítja elő. Ha az adott alkalmazásban ennél nagyobb frekvenciaintervallumra van szükség, akkor valószínűleg egy gyorsabban futó kódot eredményező nyelvet használnánk, és szándékos lassításokat helyeznénk el a teljes periódusok közé. (A lassítások fél periódusig tartanának, mert egy ON, ill. OFF művelet fél periódusig tart.) Bármilyen nyelvet is használunk, használni kell egy időtartam-számlálót is, hogy a hangot kikapcsoljuk. Egy bizonyos frekvencián úgy állíthatunk elő különböző hangokat – pl. kattanást vagy berregést –, hogy változtatjuk az impulzusok közötti késleltetést.

E csodálatos lehetőségek ellenére sem okos dolog a hangokat a hangszóró közvetlen programozásával előállítani, három súlyos hátrány miatt:

- A program állandóan a CPU teljes figyelmét igényli, ezért a számítógép nehezen tud bármi mással foglalkozni.
- A frekvencia teljes egészében a számítógép sebességétől függ: vagyis ugyanaz a program más magasságú hangot ad egy gyorsabb vagy egy lassúbb modellen.
- Az óramegszakítás zavarja a hang folytonosságát, „trillázást" okoz. Ezt csak úgy lehet elkerülni, hogy a megszakítás letiltásával felfüggesztjük az órát – ez tönkreteszi a számítógép „időérzékét".

Az ilyen hangelőállításnak tudomásunk szerint egy előnye van az időzítő használatával szemben: a késleltetések megfelelő szabályozásával gazdag polifonikus hangok állíthatók elő. Ez azonban igen ötletes és fáradságos programozást igényel, mindent összevetve mégsem éri meg a belefektetett munkát.

A hangerő és a hang minősége

A számítógép hangszórójába semmilyen hangerőszabályozó sincs beépítve. A számítógép hangszórója azonban (mint minden hangszóró) nem egyformán reagál az egyes frekvenciákra, ezért egyes frekvenciájú hangokat erősebbnek hallhatunk, mint másokat. Az olyan egyszerű hangszórónál (amilyen a legtöbb PC-ben is megtalálható) a hangerő leginkább a frekvenciától függ. A következő programmal ezt ellenőrizhetjük, és a program segíthet a célnak legjobban megfelelő hangmagasság kiválasztásában is.

```
10 PLAY "MF"                              'minden hangot külön játszik
20 FREKVENCIA=37
30 WHILE FREKVENCIA < 32000               'játssz minden frekvencián
                                          32 000 Hz-ig
40  PRINT USING "##, ###" ; FREKVENCIA    'kiírja a frekvenciát
50  SOUND FREKVENCIA,5                    '5 időegységig játszd a hangot
60  FREKVENCIA=FREKVENCIA*1.1             'növeld a frekvenciát 10 %-kal
70 WEND
```

Azzal is tisztában kell lenni, hogy az egyes PC-modellek hangszórójának eltérő a hangzása, részben azért, mert a számítógép dobozának anyagai másként rezonálnak. A PCjr hangszíne pl. nagyon eltér a Portable PC, a PC és az XT modellek hangszínétől. Gondoljunk a hang ilyen jellegű különbségeire.

Hanggenerálás a PCjr-rel

JR 

A PCjr-ben a család többi tagjához hasonlóan megtaláljuk a szabványos programozható időzítő IC-t és a beépített hangszórót, de más hangforrásai és hangkiviteli eszközei is vannak.

A PCjr legismertebb hangforrása a TI SN76496A típusú hanggenerátor IC. Ez nagyszerű eszköz játékok és oktatóprogramok sajátos hanghatásainak előállításához. A PCjr rendelkezik még két kevésbé ismert hangforrással is: a kazettás bemenettel és az audiovonallal (B30-as vonal) az I/O csatorna csatlakozóján. A négy hangforrás közül a 8255-5 típusú programozható perifériaillesztővel (PPI-vel) választhatunk. (☛ L. az 1. fejezet.) A programok a 97-es porton (hexa 61) keresztül a 8255 5. és 6. bitjének beállításával választják ki a hangforrást. (L. a 7.2. táblázatot.)

A kazettás szalag és az I/O csatorna kimenetét sztereoerősítőhöz kapcsolva a beépített hangszóróénál jobb minőségű hang hozható létre, mivel azonban ennek használata nem terjedt el, ezzel nem is foglalkozunk a továbbiakban. Vegyük inkább szemügyre a TI hang-IC-t.

7.2. táblázat
A PCjr 4 hang forrásának beállítása a 8255-A 97-es pontjának 5-ös és 6-os bitjével

Az 5-ös bit	A 6-os bit	Érték	Hangforrás
0	0	0	A 8253 időzítő (szabványos IBM hangforrás)
0	1	1	Kazettás magnó interfész
1	0	2	I/O csatorna
1	1	3	TI hang-IC

A TI hang-IC

A TI hang-IC-nek négy különböző hanggenerátora vagy hangja van. A négyből három teljesen független és tiszta hangot állít elő, akárcsak az időzítő IC. A negyedik hang zajforrás, ez többféle módon tud szabálytalan zajokat előállítani. Mind a négy hangnak függetlenül szabályozhatjuk a hangerejét. Ez 15 egyenletes beosztású hangerőszintet jelent, ill. a 0. szintet (kikapcsolt állapot). A három tiszta hang frekvenciáját külön-külön adhatjuk meg. A zajnak három kijelölt „frekvenciája" van, valamint egy negyedik lehetőség, amelyik a harmadik tiszta hangból származtatja a frekvenciát. Először egyenként foglalkozunk a hang alkotórészeivel, majd elmagyarázzuk, hogyan lehet együttes használatukkal a TI hang-IC-t programozni.

A hangmagasság előállítása

A hanggenerátorokat a programból kiküldött 10 bites számmal vezéreljük. A TI hang-IC az időzítőhöz hasonló módon állítja elő a frekvenciát: a rendszer gyors órajelét egy bizonyos számmal elosztjuk, és a hányados adja meg a frekvenciát. A részletek azonban mások. Az időzítőnél egy 16 bites számmal osztottuk el a busz 1,193 MHz-es frekvenciáját. A TI hang-IC-nél a számláló 10 bites szám, amely 1/32-es egységekben osztja le a rendszeróra 3,579 MHz-es frekvenciáját. Az egység így 111,860 Hz.

A 10 bites vezérlőszám egyetlen hátránya, hogy a választható diszkrét frekvenciák számát egy 32-es faktorral csökkenti. Ha pl. 100-as számlálót használunk, akkor ez 1118,6 Hz-nek felel meg, a következő, 101-es érték pedig 1107,5 Hz-nek. A kettő közötti frekvenciákat nem tudjuk előállítani. Ezzel ellentétben ebben az intervallumban az időzítő IC még 32 frekvenciát tudna előállítani. A gyakorlatban ez csak ritkán okoz gondot, pl. egy háromszólamú akkordban a hang esetleg hamisan szól.

A leosztás

A TI hang-IC minden hangjának külön-külön szabályozhatjuk a hangerejét. Ezt decibelben mérik. A hangerő szabályozására négy bit használható. Ezeket a biteket A0-tól A3-ig jelöljük, és segítségükkel 16 hangerőszintet lehet beállítani. Ezt illusztrálja a 7.3. táblázat. Ha egy bitet 1-re állítunk, ez bizonyos mértékben leosztja (csökkenti) a hangot: a csökkenés mértéke 2, 4,

7.3. táblázat
A leosztóbit – a hangerő beállítása

A0	A1	A2	A3	Érték	Leosztás (decibel)
.	.	.	1	1	2
.	.	1	.	2	4
.	1	.	.	4	8
1	.	.	.	8	16
1	1	1	1	32	Hangerő kikapcsolva

7.4. táblázat
Az NF0 és NF1 segítségével előállítható zajfrekvenciák

NF0	NF1	Zajfrekvencia
0	0	1 193 180 ÷ 512 = 2330
0	1	1 193 180 ÷ 1024 = 1165
1	0	1 193 180 ÷ 2048 = 583
1	1	A 3. hang

8 vagy 16 decibel. Ha mind a 4 bit értéke egy, akkor a hang ki van kapcsolva: ha pedig 0, a hang teljes erővel szól. Bár a hangerő kiszámítható, a kívánt hangot legegyszerűbb kísérletezéssel beállítani.

A zajgenerátor

A négy választható frekvencia mellett a zajt még két módon szabályozhatjuk. Az egyiket periodikus zajnak nevezzük: ez állandó hangot ad. A másikat fehér zajnak hívjuk: ez sziszegő hangot ad. A kettő között az ún. *FB*-bittel választhatunk. Ha az *FB* értéke 0, a periodikus zaj szól; ha 1, akkor fehér zaj jön létre.

Az *NF0* és *NF1* nevű bitek szabályozzák a zajgenerátor frekvenciáját. A négy kombináció közül három az időzítőtől származó önálló frekvenciát határoz meg. A negyedik kombináció frekvenciáját a harmadik, a tiszta hangot előállító hang határozza meg. A 7.4. táblázatban láthatók a zajbitek lehetséges értékei és a hozzájuk tartozó frekvenciák.

7.5. táblázat
A paraméterazonosító bitek

R0	R1	R2	Paraméter
0	0	0	Az 1. hang frekvenciáját megadó szám (10 bit)
0	0	1	Az 1. hang erőssége (4 bit)
0	1	0	A 2. hang frekvenciáját megadó szám (10 bit)
0	1	1	A 2. hang erőssége (4 bit)
1	0	0	A 3. hang frekvenciáját megadó szám (10 bit)
1	0	1	A 3. hang erőssége (4 bit)
1	1	0	Zajvezérlés (4 bit, ebből 3-at használ)
1	1	1	Zajerősség (4 bit)

A hang-IC vezérlése

A TI hang-IC 3 bites regisztercímek kiküldésével (ezek határozzák meg a hangok frekvenciáját, a leosztást és a zaj hangját), 10 bites frekvencialeosztásokkal, 4 bites leosztásbeállításokkal, 2 bites zajfrekvencia beállításával és egy 1 bites zajtípus kiválasztásával programozható. Mindez a 192-es (hexa C0) porton megy keresztül. Nem könnyű mindezt belesűríteni egy 8 bites portba!

A TI hang-IC működését 8 belső regiszter vezérli. Az *R0, R1, R2* nevű 3 bit választja ki a regisztert, és azonosítja a beállítandó paramétert. A regiszterértékek a 7.5. táblázatban láthatók.

A TI hang-IC programozása teljes egészében a 192-es portra írás. Ne olvassuk ki a portot, mert leblokkoljuk a gépet! Ez a bájt tartalmazza az *ID*-bitet (ez jelzi, hogy parancsbájtról van szó), a regisztereket jelölő *R0, R1, R2* biteket és néhány adatbitet. A parancsbájt segítségével lehet betölteni a frekvenciát, a leosztást, ill. a zajt szabályozó biteket és a regiszterek bitjeit. Az adatbitek formátuma jelentésüktől függően más és más. A tiszta hangok frekvenciáját megadó számlálóértékek esetén az első bájtot követi az 1 db *ID*-bitet tartalmazó 2. bájt, amelyben a 6 frekvenciabitet is elhelyezték. A tervezők szándékosan úgy jártak el, hogy a 2. bájt az 1. bájt megismétlése nélkül is ismételten kiküldhető legyen. Ez lehetővé teszi, hogy a frekvenciát gyorsan megváltoztassuk, és elkerüljük azt a veszteséget, amit mindkét bájt betöltése okozna. A 7.6., a 7.7., a 7.8. és a 7.9. táblázatokban látható az egyes bájtok bitformátuma.

7.6. táblázat
A frekvenciaszámláló 1. bájtjában szereplő bitek jelentése

Bit 7	6	5	4	3	2	1	0	Jelentése
1	.	.	.	.	.	.	.	Az 1. bájtot azonosítja (parancsbájt)
.	R0	R1	R2	.	.	.	.	A regiszter száma a TI IC-ben (0,2 vagy 4)
.	.	.	.	F6	F7	F8	F9	A frekvenciát megadó 10 bitből 4

7.7. táblázat
A frekvenciaszámláló 2. bájtjában szereplő bitek jelentése

Bit 7	6	5	4	3	2	1	0	Jelentése
0	.	.	.	.	.	.	.	A 2. bájtot azonosítja (a szám kiegészítője)
.	X	.	.	.	.	.	.	Nincs felhasználva: lehet akár 0, akár 1
.	.	F0	F1	F2	F3	F4	F5	A frekvenciát megadó szám 10 bitjéből 6

7.8. táblázat

A frekvenciaszámláló 1. bájtjában a hangerőt szabályozó bitek jelentése

Bit 7	6	5	4	3	2	1	0	Jelentése
1	.	.	.	.	.	.	.	Az 1. bájtot azonosítja (parancsbájt)
.	R0	R1	R3	.	.	.	.	A regiszter száma a TI IC-ben (1, 3, 5 vagy 7)
.	.	.	.	A0	A1	A2	A3	A hangerő 4 bitje

7.9. táblázat

A frekvenciaszámláló 1. bájtjában a zajt vezérlő bitek jelentése

Bit 7	6	5	4	3	2	1	0	Jelentése
1	.	.	.	.	.	.	.	Az 1. bájtot azonosítja (parancsbájt)
.	1	1	0	.	.	.	.	A TI IC regiszterének száma (6)
.	.	.	.	X	.	.	.	Nincs felhasználva: lehet akár 0, akár 1
.	.	.	.	.	FB	.	.	1 fehér zajnál, 0 periodikusnál
.	.	.	.	.	.	NF0	NF1	A zajfrekvenciáját szabályozó 2 bit

📖 MEGJEGYZÉS: Mielőtt valaki hozzákezd a TI hang-IC programozásához, jó ha ismeri a Texas Instruments és az Intel tervezési eljárása közötti bosszantó különbséget: az eltérően jelölt bitsorrendet. Ennek az a következménye hogy a PCjr Technical Reference kézikönyvben a hang-IC bitjeire ellenkező sorrendben hivatkoznak, mint a kézikönyv más részeiben. (Ez a könyv végig a szokásos jelölést használja.) Azt a bitet, amit máshol egy bájt 7-es bitjének neveznek, a hang-IC leírásánál 0-s bitnek vagy MSB-nek (Most Significant Bit, legmagasabb helyiértékű bit) nevezik. A hivatkozási kézikönyvben jobb az MSB vagy az LSB jelölést használni, és eltekinteni a bitek számozásától. (Közben bosszankodhatunk a jelölési mód felelőtlen felcserélése miatt.)

Ebből az áttekintésből is biztosan kitetszik, hogy a TI hang-IC programozása rendkívül bonyolult. Valahányszor megpróbálkozunk vele, mindig arra kényszerülünk, hogy folyton a biteket számoljuk. További hátrány, hogy a hang-IC-t használó programok csak a PCjr-en érvényesülnek igazán, és nem zenélnek a PC-család többi tagján. Ezt feltétlenül figyelembe kell venni, mielőtt belekezdenénk a hang-IC programozásába!

Hangkimenet a PCjr-en

A PCjr hangjeleinek két végállomása van: a beépített hangszóró és a külső utak. A külső utak a PCjr hátoldalán található csatlakozók: az *A* (audiokimenet), ezt rendszerint hi-fi rendszerhez csatlakoztatják; a *D* (közvetlen meghajtású kimenet), ez pl. egy RGB-monitorhoz csatlakoztatható, amely rendszerint figyelmen kívül hagyja a hangot; végül a *T* (televíziókimenet), ez az RF-modulátorral egy tv-készülékhez csatlakoztatható.

A PCjr beépített hangszórójára (mint az összes PC belső hangszórójára) csak az időzítőből származó hangjel kerülhet. A külső csatlakozókra mind a négy lehetséges hangforrásból kerülhet jel. Fontos szempont, hogy a TI hang-IC hangját nem lehet a beépített hangszóróba vezetni, mert a beépített hangszóró meghajtása teljesen alkalmatlan arra, hogy kihasználja a TI hang-IC által nyújtott gazdag lehetőségeket.

A PCjr Technical Reference kézikönyv tanulmányozásakor rá lehet bukkanni, hogy a 97-es port 4. bitje felhasználható a beépített hangszóró vezérlésére. Ha ezt a bitet 1-re állítjuk, akkor letiltjuk a hangszórót. Mégse tegyük ezt! Egyrészt van más mód is a hangszóró vezérlésére, másrészt (és ez sokkal fontosabb) a többi PC-modell ezt a bitet egészen másra használja: ha ez a bit 1, akkor letiltja a tár használatát, ez pedig alaposan leblokkolja a gépet.

8 A ROM-BIOS alapjai

Bevezetés • 138

A BIOS filozófiája • 138

A ROM-BIOS-szolgáltatások megszakításai • 139

A BIOS-szolgáltatások működésének jellemzői • 139

Egy assembly nyelvű interfész létrehozása • 140

Egy interfész szubrutin kerete • 141

Bevezetés

A PC-család sikeres programozásának egyik titka a gépbe beépített szoftver – az ún. ROM-BIOS-szolgáltatások – használata. Elvileg a ROM-BIOS szolgáltatásai (ideértve az operációs rendszert) a hardver és a magasszintű nyelvek közé szendvicsszerűen ékelődnek be. Közvetlenül a hardverrel és a perifériákkal állnak kapcsolatban, és a rendszer alapvető feladatait látják el: pl. az egyes adatbájtok beolvasását, ill. kiírását a képernyőre vagy a lemezre. A programozási nyelvek és a DOS szolgáltatásai gyakran ezekből az elemi funkciókból épülnek fel, hogy egyes folyamatokat hatékonyabbá tegyenek. Saját programjainkat is javíthatjuk, ha ROM-BIOS-ba ültetjük őket, ezzel felhasználhatunk egy hatásos eszköztárat, úgy kezelve a számítógépet, ahogy az IBM elgondolta.

Az utolsó részt érdemes hangsúlyozni. Az IBM nagy erőfeszítéseket tett azért, hogy a ROM-BIOS-szolgáltatásokkal a számítógépet világos, jól definiált módon irányíthassuk. Minden egyes új PC-modell tervezésekor az IBM (de minden más számítógépgyártó is, aki az eredeti elvekkel összhangban bővíti a PC-családot) meggyőződik arról, hogy a ROM-BIOS szolgáltatásai teljesen kompatíbilisek a család többi tagjával. Míg a számítógépet (közvetlenül vagy közvetve) a ROM-BIOS segítségével irányítjuk, biztosan védve vagyunk minden kompatibilitási gondtól. Ha viszont közvetlenül a hardvert programozzuk, megnehezítjük a munkánkat, és erősen korlátozzuk azon gépek körét, amelyekkel programunk használható lesz.

A most következő öt fejezetben a ROM-BIOS szolgáltató szubrutinjait ismertetjük. Szerencsére a szolgáltatások természetes csoportokra oszthatók aszerint, hogy melyik hardvereszközt szolgálják ki. Ezért külön-külön tekinthetjük át a képernyős szolgáltatásokat, a lemezes szolgáltatásokat és a billentyűzet szolgáltatásait. Mielőtt azonban az egyes csoportok részletes ismertetésébe kezdenénk, először azt kell megbeszélnünk, hogy lehet ezeket a szolgáltatásokat programjainkba építeni. Elmagyarázzuk, miből áll a programozási nyelveket és a ROM-BIOS-szolgáltatásokat összekötő interfész szubrutin, legelőször azonban a ROM-BIOS működésével foglalkozunk.

A BIOS filozófiája

Minden ROM-BIOS-szolgáltatást megszakítással hívunk meg. A megszakítási utasítás a megszakítási vektorok táblázatának megadott címére mutat. Ez a táblázat a tár alacsony című részén található, és a megszakítási vektorok címét (valójában a kiszolgáló szubrutin tárbeli címét) tartalmazza. Ez a tervezés lehetővé teszi, hogy bármelyik program igénybe vegyen egy szolgáltatást anélkül, hogy ismerné az adott szubrutin tényleges tárcímét. Egy-egy szolgáltatás így elmozdítható, bővíthető vagy adott körülményekhez igazítható a szolgáltatást használó programok megzavarása nélkül. Bár az IBM megpróbálja a ROM-BIOS egyes részeinek abszolút tárcímét változatlanul megőrizni, rosszul járnánk, ha ezt a címet használnánk, hiszen a változtatás veszélye mindig fennáll. Egy ROM-BIOS-szolgáltatás meghívásának szabványos és legmegbízhatóbb módja az, ha a megszakítással hívjuk meg – nem pedig abszolút tárcíme segítségével.

A ROM-BIOS szolgáltatásaira egyetlen „fő" megszakítás is felügyelhet; ehelyett azonban összefüggő csoportokra lettek felosztva, és minden csoportnak megvan a saját vezérlőmegszakítása. A tervezés elsődleges előnye az, hogy a megszakítások kezelői a lehető legkisebb változtatással cserélhetők. Ha pl. egy hardvergyártól egy teljesen eltérő képernyőt, nyomtatót vagy bármi mást állítanak elő, aminek a működtetése is új BIOS-programot igényelne, akkor

ez a gyártó mellékelhetné a hardverhez az új BIOS-t is. Az új BIOS valószínűleg nem ROM-ban, hanem RAM-ban lenne, és az IBM a ROM-BIOS-nak csak azt a részét változtatná meg, amelyik az új hardverhez kell. A ROM-BIOS moduláris tervezésével az IBM megkönnyítette számítógépeinek javítását és bővítését.

A ROM-BIOS-szolgáltatások megszakításai

Összesen 12 ROM-BIOS-megszakítás van, és ezek 5 csoportra oszthatók. A 12 megszakításból 6 a megadott perifériát kezeli; 2 a számítógéphez csatlakoztatott berendezések jelenlétét vizsgálja; 1 az órával és a dátummal foglalkozik; 1 a képernyő tartalmát nyomtatja; és végül 2 a számítógép működésének más területéhez kapcsolódik: a ROM-BASIC-et, ill. a bekapcsolási szubrutint indítja. Látni fogjuk, hogy a legtöbb megszakítás az alárendelt szolgáltatások egy csoportjára épül, amelyek majd elvégzik a munka lényegét. A képernyős szolgáltatások megszakításaihoz (16-os megszakítás) 16 alárendelt szolgáltatás kapcsolódik. Ezek – a megjelenítési mód beállításától a kurzor méretének megváltoztatásáig – mindent elvégeznek. Egy alárendelt szolgáltatást a vezérlőmegszakítással hívunk meg, és az *AH*-regiszterben adjuk meg az alárendelt szolgáltatás számát. (☛ A fejezet végén ezt egy példán elmagyarázzuk.)

A BIOS-szolgáltatások működésének jellemzői

A ROM-BIOS-szolgáltatások hívási szabályai következetesek a regiszterek, a jelzők, a verem és a tár használatában. A hívási szabályok jellegzetességeinek ismertetését a szegmensregiszterek használatával kezdjük.

A *CS*-regiszter (a kódszegmens) tartalmának kimentése, feltöltése és visszamentése a megszakítási folyamat része. Ennek következtében nem kell foglalkozni programjaink *CS*-regiszterével. A *DS*- és *ES*-regisztereket a ROM-BIOS szolgáltató szubrutinjai megőrzik, kivéve azt a néhány esetet, ahol explicit módon felhasználják. A verem szegmensregisztere (az *SS*) tartalma nem változik, a ROM-BIOS-szolgáltatások ránk bízzák, hogy elég nagy vermet biztosítsunk. (A verem méretén nagyon sok múlik!)

A ROM-BIOS-szolgáltatások veremigényét nem tudhatjuk meg, mert folyton-folyvást változik, mivel egyes szolgáltatások behívnak másokat. (Az IBM személyi számítógépek egyik apró hibája – valójában az Intel 8088-at használó összes számítógépnek is hibája – a verem bonyolult használata és a veremhatárok zavaros specifikációja.) Általában a legtöbb programnak jóval nagyobb vermet kell biztosítani, mint amit a ROM-BIOS-szolgáltatások igényelnek.

A szegmensregiszterekhez kapcsolódva kell elmondani, hogy az utasításszámlálót (*PC* vagy *IP*) ugyanaz a mechanizmus őrzi meg, mint a szegmensregisztert. Lényegében a veremmutató (*SP*) is megtartja értékét, mert minden ROM-BIOS-szolgáltatás kiüríti a vermet; azaz mindent kivesz belőle, amit a szubrutin végrehajtása közben beletett.

Az általános célú regiszterek a szokásos módon (*AX*-tól *DX*-ig) szabadon felhasználhatók. Általános szabály: ne várjuk el, hogy ezek a regiszterek megőrzik értéküket, ha a vezérlést másik szubrutinnak adjuk át. Ez érvényes a ROM-BIOS-szolgáltatásokra is. Ha valaki alaposan megnézi a szolgáltatások kódját az IBM Technical Reference kézikönyvben, akkor megfigyelheti, hogy egyik-másik nem rontja el valamelyik regiszter tartalmát. Ezt azonban meggondolatlanság lenne kihasználni. Általános szabály az is, hogy ha egy szubrutin egyszerű ered-

8.1. táblázat
A 12 ROM-BIOS-szolgáltatás

Megszakítás Decimális	Megszakítás Hexadecimális	Jelentése
Perifériákkal kapcsolatos szolgálatások		
16	10	Videoképernyő-szolgáltatások (l. 9. fejezet)
19	13	Lemezes szolgáltatások (l. 10. fejezet)
20	14	Kommunikációs szolgáltatások (l. 12. fejezet)
21	15	Kazettás magnetofonnal kapcsolatos szolgáltatások (l. 12. fejezet)
22	16	Szabványos billentyűzetszolgáltatások (l. 11. fejezet)
23	17	A nyomtatóval kapcsolatos szolgáltatások (l. 12. fejezet)
A berendezések állapotát megadó szolgáltatások		
17	11	A berendezések listáját megadó szolgáltatás (l. 12. fejezet)
18	12	A tárméretet megadó szolgáltatás (l. 12. fejezet)
Idő és dátum szolgáltatás		
26	1A	Idő- és dátumszolgáltatások (l. 12. fejezet)
A képernyő tartalmát kinyomtató szolgáltatás		
5	5	A képernyő tartalmát kinyomtató szolgáltatás (l. 12. fejezet)
Speciális szolgáltatások		
24	18	A ROM-BASIC indítása (l. 12. fejezet)
25	19	Az öntöltő betöltőprogram indítása (l. 12. fejezet)

ménnyel tér vissza, akkor az az *AX*-regiszterben található. Ez a ROM-BIOS-ra és valamennyi programozási nyelvre vonatkozik. Látni fogjuk, hogy ez milyen gyakori, ha majd részletesen foglalkozunk a ROM-BIOS-szolgáltatásokkal.

Az indexregiszterek (*SI* és *DI*) éppúgy megváltoztathatók, mint az *AX*–*DX*-regiszterek. A jelzőregiszter különféle jelzői a ROM-BIOS-utasítások végrehajtásával változnak. Nem várhatjuk el, hogy valamelyik változatlan maradjon. Néhány esetben az átviteljelző *(DF)* vagy a zérusjelző *(ZF)* segítségével értesülünk a művelet sikeréről.

Ezek a részletek fontosak, de meglehetősen unalmasak. Ha valaki betartja a következő részben ismertetendő interfészek általános szabályait és az egyes programozási nyelvek által támasztott igényeket, akkor egyáltalán nem is kell velük törődni. (☛ Erről részletesen a 19. és a 20. fejezetben lesz szó.)

🕮 MEGJEGYZÉS: Ha valaki arra vállalkozik, hogy programjában felhasználja a ROM-BIOS-szolgáltatásokat, akkor természetesen aggódni kezd, hogy konfliktus jöhet létre a szolgáltatások és a nyelv működési konvenciói között. Legyünk nyugodtak, rá fogunk jönni, semmilyen rendkívüli óvintézkedést sem kell tennünk ahhoz, hogy megvédjük a programozási nyelvet a ROM-BIOS-tól és viszont.

Egy assembly nyelvű interfész létrehozása

A ROM-BIOS szolgáltatásainak közvetlen felhasználásához létre kell hozni egy assembly nyelvű interfész szubrutint, ami kapcsolatot teremt a programozási nyelv és a ROM-BIOS között. Interfész szubrutinon olyan hagyományos programfejlesztői szubrutint értünk, amelyet

tárgymodulra fordítunk le. (.OBJ kiterjesztésű fájlra), majd működő programmá szerkesztünk össze, ami a DOS terminológiájával. .EXE vagy .COM kiterjesztésű fájl. (☛ Ezzel a témával részletesebben a 19. fejezet foglalkozik).

Az assembly nyelv félelmetesnek tűnhet annak, aki nem foglalkozott még vele. Minden okunk megvan rá, hogy féljünk az assembly nyelvtől – hiszen ez a legbonyolultabb, legigényesebb programozási nyelv –, egy assembly nyelvű interfész szubrutin megírása mégsem annyira nehéz. Amikor a szerző először írt interfész szubrutint a Norton Utilities programokhoz, magukat a programokat Pascalban írta, semmilyen előzetes gyakorlata sem volt az IBM Assemblerrel, ill. a 8088 gépi vagy assembly nyelvével. Bár nulláról indult, 45 percen belül elkészült az első kipróbált és működő interfész. Ebből is látszik, hogy a feladat nem is olyan nehéz, mint ahogy egyesek gondolják.

Az interfész készítéséhez szükség lesz egy olyan assemblerre, amelyik a DOS-szal kompatíbilis tárgykódú fájlokat állít elő. A szerző az IBM Macro Assemblert használja, de mások is megfelelnek. Semmi esetre sem javasoljuk David Whitman joggal híres „olcsó assemblerét", a CHASM-ot. A CHASM és néhány más assembler csak teljes, futásra kész programok előállítására van felkészülve, nem pedig más programokhoz szerkeszthető modulokéra; mint pl. a számunkra érdekes interfész modulok. A közölt példák mind az IBM Macro Assemblerrel készültek.

🕮 MEGJEGYZÉS: A BASIC közvetlenül is képes együttműködni a tárba tett gépi kódú szubrutinokkal. Ez az interpreteres BASIC-ben **CALL**-lal hívható, a fordítós BASIC-ben pedig a **CALL** abszolút formájával. A BASIC-kel együttműködő assembly szubrutint elkészíthetjük akár a DEBUG **A**-parancsával, akár egy közönséges assemblerrel.

Egy interfész szubrutin kerete

Az interfész szubrutin pontos alakja függ a tényleges felhasználástól. Az assembly nyelvű interfész szubrutin összekötő láncszem a programozási nyelv és a ROM-BIOS-szolgáltatás között. Úgy kell tehát kialakítani, hogy mindkét fél igényeit kielégítse. Fontos, hogy milyen programozási nyelvről van szó; melyik ROM-BIOS-szolgáltatást hívjunk; és van-e adatátadás valamelyik irányban. Ennek ellenére (függetlenül a céljától) minden assembly nyelvű interfész váza alapvetően megegyezik.

Talán úgy lehet legjobban megérteni egy assembly nyelvű interfész kódolását, ha az alábbi vázlat szerint egy ötszörösen egymásba ágyazott struktúrának tekintjük:

1. szint: az általános rész
 2. szint: a szubrutinhoz tartozó rész
 3. szint: a belépési kód
 4. szint: a paraméterek átvétele a hívótól
 5. szint: a ROM-BIOS-szolgáltatás meghívása
 4. szint: az eredmények visszaadása a hívónak
 3. szint: a kilépési kód
 2. szint: a szubrutinhoz tartozó rész lezárása
1. szint: az általános rész lezárása

A vázlatban az 1. és 2. szint azt közli az assemblerrel, hogy mi is történik, de nem hoz létre végrehajtható utasításokat. A 3. és 5. szint közötti részek állítják elő a tényleges gépi kódú utasításokat.

Át fogjuk rágni magunkat az összes rétegen, miközben felhívjuk a figyelmet a szabályokra, és a folyamatokra. Ne felejtsük, hogy az interfész szubrutinokkal szemben támasztott követelmények a körülmények függvényében változnak. Hangsúlyozzuk azt a néhány tervezési szempontot, amelyik minden szubrutinnál érvényes.

1. szint: az általános rész

A következő példában egy interfész szubrutin jellegzetes részlete látható az 1. szinten (a sorokat megszámoztuk, hogy könnyebben hivatkozhassunk rájuk):

```
1.1. INTERFACE SEGMENT 'CODE'
1.2.          ASSUME CS:INTERFACE
     ; ide kerül a 2 – 5. szint
1.3. INTERFACE ENDS
1.4.          END
```

Az 1.1. sorban álló **INTERFACE** az assembly szubrutin általunk választott neve; a **SEGMENT** szót pontosan így kell használni, assembly nyelvű szubrutin kezdetét jelzi; a **CODE** nyelvtől függő kategóriát jelez (hamarosan más példát is fogunk látni).

Az 1.2. sorra nincs mindig szükség, az assemblerek ismerői rájönnek, hogy szabálytalan dologról van szó. Az **ASSUME**-utasítás egyszerűen „ha-akkor" típusú programozást tesz lehetővé anélkül, hogy bonyodalmakat okozna; a későbbi fejezetekben még használni fogjuk.

Az 1.3. sor lezárja a megkezdett szegmenst.

Az 1.4. sor pedig az egész assembly szubrutint lezárja.

Az imént bemutatott formai konvenciók az IBM Microsoft Pascalból származnak (☛ Ezzel még foglalkozunk a 20. fejezetben.) Egy másfajta példában az 1.1. sor helyett a C nyelv ezt a két sort igényli:

```
PGROUP          GROUP          PROG
     INTERFACE SEGMENT BYTE PUBLIC "PROG"
```

2. szint: a szubrutinhoz tartozó rész

Nézzünk egy jellegzetes példát a 2. szintre! Ezt a szubrutinhoz tartozó assembler-résznek (assembler dialektusban eljárásnak) nevezik. Tehát a 2. szint tipikus kódja:

```
2.1.                    PUBLIC     MEMSIZE
2.2. MEMSIZE            PROC       FAR
     ; ide kerül a 3 – 5. szint
2.3. MEMSIZE            ENDP
```

A 2.1. sor arra utasítja az assemblert, hogy az eljárás nevét – **MEMSIZE** – kezelje nyilvános adatként, tehát a szerkesztőprogram más felhasználók programjához is hozzá tudja szerkeszteni.

A 2.2. és a 2.3. sorok határolják eljárásunkat, amelynek ötletszerűen adtuk a **MEMSIZE** nevet. A **PROC** és az **ENDP** használata kötelező, ezek határolják az eljárást. A **PROC** jelzi az eljárás kezdetét, az **ENDP** a végét. A **FAR** azt közli az assemblerrel, hogy az eljárás az aktuális

szegmensen kívül található. Ezen a helyen **FAR** vagy **NEAR** állhat. Ha **NEAR**-t írtunk volna, az azt jelenti, hogy az eljárás az aktuális szegmensen belül, nem pedig kívül található. A **FAR** típusú hívások a legáltalánosabbak (l. a Pascalt), néhány nyelv azonban **NEAR** típusú hívást használ (pl. a C). A **FAR** és a **NEAR** kivételével a többi itt látható elem minden nyelvnél és minden célra általánosan használható.

3. szint: a belépési kód

A 3. szinttel kezdődnek az igazi végrehajtható utasítások. Ez tartalmazza azt az adminisztrációt, ami biztosítja, hogy a szubrutin együttműködhessen az őt meghívó programnyelvvel. Íme egy példa:

```
3.1.   PUSH    BP
3.2.   MOV     BP,SP
       ;ide kerül a 4–5. szint
3.3.   POP     BP
3.4.   RET     0
```

A 3.1. és a 3.2. sor azt teszi lehetővé, hogy a hívó által átadott paraméterekhez hozzá lehessen férni, és azok megőrződjenek. Ezt segíti valamilyen módon a verem. A bázismutató regiszterének *BP* használata általános, a verem belépésekor érvényes állapotának megőrzésére. A hívóprogramnak saját *BP*-regiszter értéke van, ezt a 3.1. sorban mentjük azzal, hogy a verembe tesszük, majd a 3.3. sorban az eredeti tartalmát helyreállítjuk.

A 3.2. sortól kezdve a saját veremre hivatkozunk, mivel az aktuális veremmutatót *(SP)* a *BP*-regiszterbe tettük. Ettől kezdve a verem további használatától függetlenül megjegyeztük, hogy hol vannak a hívó paraméterei a veremben. Ha a hívó bizonyos regisztereinek tartalmát meg kellene őriznünk, akkor ezeket közvetlenül a 3.2. utasítás után kellene a verembe tenni, és közvetlenül a 3.3. sor után – fordított sorrendben – kellene onnan kivenni őket. Általában erre nincs szükség.

A 3.4. sor a hívónak adja vissza a vezérlést. Az assembler az egyszerű **RET**-utasítást **NEAR** vagy **FAR** típusú visszatérésre fordítja, attól függően, hogy a **PROC NEAR** vagy **FAR** kiegészítésű-e. A 3.4. sorban álló 0 a rendrakást szolgálja; a hívó paramétereit távolíthatjuk el vele a veremből. Ha nem voltak paraméterek, vagy ha a programozási nyelv konvenciói a hívó kötelességévé teszik a verem ürítését (pl. ilyen a C nyelv, ☛ l. a 2. fejezetet), akkor ez az érték 0. Ha voltak paraméterek, és a programozási nyelv nem üríti a vermet, akkor meg kell adni ezt az értéket, hogy a paraméterektől a verem megszabaduljon! A számhoz kettőt kell adni minden 1 és 2 bájtos paraméternél (bájt, szó, relatív cím), és négyet a 4 bájtos paraméterek (szegmentált cím) esetében. Minden rendben lesz, ha ismerjük az egyes paraméterek típusát, mert az biztosan a fenti négyből kerül ki.

4. szint: a paraméterek átvétele a hívótól

A 4. szint a paraméterek átadásával foglalkozik: a hívótól a ROM-BIOS felé továbbítja a paramétereket, az eredményeket pedig a ROM-BIOS felől a hívónak. A hívó paraméterei a veremben találhatók, vagy adatok, vagy címek formájában. (☛ A 20. fejezetben ezzel még foglalkozunk). A ROM-BIOS a regisztereket – legtöbbször az *AX – DX*-regisztereket – használja az adatok átvételére és az eredmény közlésére. Itt arra kell figyelni (erre nagyon

vigyázzunk), hogy a vermen belül a helyes relatív címet használjuk a paraméter kivételekor. Ennek a megoldását több lépésben közelítjük meg.

Először is a veremben levő paraméterhez úgy férünk hozzá, hogy az előbb már kimentett *BP*-regiszterhez képest relatív módon címezünk. Egy jellegzetes példa:

Cím	Tartalom
BP	A hívó kimentett *BP*-tartalma
BP+2	Visszatérési cím relatív és szegmens része
BP+6	Az egyik paraméter
BP+8	A másik paraméter
BP+10	A harmadik paraméter

A *BP*+2-n található visszatérési cím **FAR**-eljárásoknál 4 bájt, **NEAR**-eljárásoknál pedig 2 bájt. Ha **NEAR**-eljárással van dolgunk, akkor az ezt követő relatív címek a bemutatottnál kettővel mindig kisebbek. A legtöbb nyelv a paramétereit a leírás sorrendjében teszi a verembe. Tehát az utolsó paraméter van legközelebb a verem tetejéhez, a *BP*+6 címen. (A Lattice/Microsoft C azonban fordított sorrendet használ, tehát a legközelebbi paraméter a hívóprogramban elsőnek leírt paraméter).

A paraméterek általában 2 vagy 4 bájtot foglalnak el a veremben, a 2 bájt az általánosabb. Példánkban a *BP*+6, *BP*+8, *BP*+10 két-két bájtnyira van egymástól. Ha valamelyik paraméter 4 bájtos lenne, az őt követő hivatkozásokat megfelelően módosítani kellene.

Ha adat van a veremben, akkor azt egy *BP*+6 jellegű címzéssel rögtön kivehetjük. Ha cím van a veremben, akkor az adatot két lépésben érhetjük el: először kivesszük a címet, aztán megkapjuk az adatot. Íme egy példa a 4. szintre adat (*BP*+6) és cím (*BP*+8) visszakereséssel:

```
4.1.    MOV    AX, [BP+6]          ; az 1. paraméter értéke
4.2.    MOV    DX, [BP+8]          ; a 2. paraméter címe
4.3.    MOV    BX, [DX]            ; a 2. paraméter értéke
        ; ide kerül az 5. szint
4.4.    MOV    DX, [BP+8]          ; ismét a 2. paraméter címe
4.5.    MOV    [DX], BX            ; az új érték visszaadása
```

Az összes **MOV**-utasítás a 2. operandusból az 1-be mozgatja az adatot. A 4.1. sor közvetlenül a veremből veszi ki az adatot, és az *AX*-regiszterbe teszi. A 4.2. és a 4.3. sorok a veremben tárolt cím segítségével jutnak az adathoz; a 4.2. sor előveszi a címet és a *DX*-regiszterbe viszi, majd a 4.3. sor ezen cím felhasználásával fér hozzá a tényleges adathoz, és azt a *BX*-regiszterbe teszi. A 4.4 és a 4.5 sorok megfordítják a folyamatot: a 4.4 sor ismét a címet veszi elő, aztán a 4.5 sor a *BX* tartalmát ezen a tárcímen helyezi el.

📖 MEGJEGYZÉS: A példa az assemblyben használt jelölés egyik kulcsfontosságú elemét is szemlélteti: a *DX* azt jelöli, ami a *DX*-ben van, a [DX] arra a tárrekeszre hivatkozik, amelyiknek a címe a *DX*-ben van.

Nem állíthatjuk, hogy ezeknek a hivatkozásoknak az alkalmazása bagatell dolog lenne, de ha pontosan átgondoljuk, akkor hibátlanul kell működnie.

5. szint: a ROM-BIOS-szolgáltatás meghívása

Az 5. szint az utolsó lépés: egyszerűen meghív egy ROM-BIOS-szolgáltatást. Ez a lépés rendszerint két egyszerű utasításból áll:

```
5.1.    MOV    AH,1    ; 1-es funkció
5.2.    INT    16      ; BIOS-szubrutin hívása
```

Az 5.1. sor választja ki a megszakításon belüli alárendelt szolgáltatást. Rendszerint több alárendelt szolgáltatás is van, ezeket 0-tól kezdve számozzák, és mindig az *AH* -regiszterrel választják ki.

Az 5.2. sor a megszakítást generálja, ez indítja a szolgáltatást. Példánkban ez a 16-os video-szolgáltatás megszakítása.

Ez az ötlépéses folyamat az assembly nyelvű interfész szinte minden alapelvét bemutatta. A következő fejezetekben a módszer konkrét használatát láthatjuk.

9 A ROM-BIOS képernyős szolgáltatásai

Bevezetés • 148

A BIOS képernyős szolgáltatásainak elérése • 148

A 0-s szolgáltatás: a megjelenítési mód beállítása • 149
Az 1-es szolgáltatás: a kurzor méretének beállítása • 149
A 2-es szolgáltatás: a kurzor pozíciójának beállítása • 150
A 3-as szolgáltatás: a kurzor pozíciójának kiolvasása • 151
A 4-es szolgáltatás: a fényceruza pozíciójának kiolvasása • 151
Az 5-ös szolgáltatás: az aktív képernyőoldal beállítása • 152
A 6-os szolgáltatás: az ablak görgetése felfelé • 152
A 7-es szolgáltatás: az ablak görgetése lefelé • 153
A 8-as szolgáltatás: karakter és attribútumának kiolvasása • 154
A 9-es szolgáltatás: karakter és attribútumának kiírása • 154
A 10-es szolgáltatás (hexa A): karakterkiírás • 155
A 11-es szolgáltatás (hexa B): a színpaletta beállítása • 156
A 12-es szolgáltatás (hexa C): a képpont kigyújtása • 157
A 13-as szolgáltatás (hexa D): a képpont kiolvasása • 157
A 14-es szolgáltatás (hexa E): karakterkiírás teletype módban • 158
A 15-ös szolgáltatás (hexa F): az aktuális megjelenítési mód kiolvasása • 159
A 19-es szolgáltatás (hexa 13): a karakterlánc kiírása • 159

Megjegyzés és példa • 159

Bevezetés

Ebben a fejezetben a ROM-BIOS által biztosított összes képernyővezérlő szolgáltatást ismertetjük. A fejezet legnagyobb részét az egyes videoszolgáltatások ismertetésének szenteltük. A fejezet végén néhány programozási tanácsot adunk, és közlünk egy assembly nyelvű szubrutint, amely néhány videoszolgáltatást használ fel. (☛ A video jellegzetességeit általánosabban a 4. fejezetben ismertetjük. A ROM-BIOS által video-státuszinformációk tárolására használt tárcímekkel kapcsolatban a 3. fejezetben találhatók adatok.)

A BIOS képernyős szolgáltatásainak elérése

A ROM-BIOS képernyős szolgáltatásait a 16-os megszakítással lehet kérni. Ehhez a megszakításhoz 16 általános szolgáltatás és egy AT-szolgáltatás tartozik (l. a 9.1. táblázatot). Mint az összes ROM-BIOS-szolgáltatásnál, a számozás itt is 0-tól indul, és a szolgáltatást a megfelelő szám *AH*-regiszterbe helyezésével választhatjuk ki. A szolgáltatások gyakran kérnek kiegészítő paramétereket is a hívótól, ezeket a specifikációnak megfelelően a *BX*-, a *CX*-, ill. a *DX*-regiszterbe kell tenni. Az egyes paraméterek feladatát és elhelyezését az egyes szolgáltatások leírásánál ismertetjük.

9.1. táblázat
A 17 videoszolgáltatás

Szolgáltatás		Jelentése
Decimális	Hexadecimális	
0	0	A megjelenítési mód beállítása
1	1	A kurzor méretének beállítása
2	2	A kurzor pozíciójának beállítása
3	3	A kurzor pozíciójának kiolvasása
4	4	A fényceruza pozíciójának kiolvasása
5	5	Az aktív képernyőoldal beállítása
6	6	Az ablak görgetése felfelé
7	7	Az ablak görgetése lefelé
8	8	Karakter és attribútum olvasása
9	9	Karakter és attribútum írása
10	A	Karakter írása
11	B	Színpaletta beállítása
12	C	Képpont kigyújtása
13	D	Képpont kiolvasása
14	E	Karakter írása teletype módban
15	F	Az aktuális megjelenítési mód beolvasása
19	13	Karakterlánc kiírása

A 0-s szolgáltatás: a megjelenítési mód beállítása

A 0-s szolgáltatással lehet a 9.2. táblázatban látható megjelenítési módok közül a megfelelőt beállítani. (☛ A megjelenítési módokat részletesen a 4. fejezetben ismertetjük.)

Emlékeztetünk rá, hogy a 4. fejezetben leírtak szerint, a 0–6-os módok a szabványos színes-grafikus adapterhez tartoznak, a 7-es módot a monokrom adapterhez, a 8-as, 9-es és 10-es módot a PCjr-hez találták ki, a 13-as, 14-es és 15-ös módot pedig az EGA-kártyánál vezették be. Az EGA-adapter a 8-as és 9-es mód kivételével az összes megjelenítési módot támogatja.

9.2. táblázat
A megjelenítési módok beállítása a 0-s szolgáltatásnál

Mód	Típus	Színek	Felbontás	Méretek	Adapter
0	Szöveges	Fekete-fehér	Közepes	40 × 25	CGA
1	Szöveges	16	Közepes	40 × 25	CGA
2	Szöveges	Fekete-fehér	Nagy	80 × 25	CGA
3	Szöveges	16	Nagy	80 × 25	CGA
4	Grafikus	4	Közepes	320 × 200	CGA
5	Grafikus	4 szürke	Közepes	320 × 200	CGA
6	Grafikus	Fekete-fehér	Nagy	640 × 200	CGA
7	Szöveges	Fekete-fehér	Nagy	80 × 25	MA
8	Grafikus	16	Kicsi	160 × 200	PCjr
9	Grafikus	16	Közepes	320 × 200	PCjr
10	Grafikus	4,64	Nagy	640 × 200	PCjr, EGA
13	Grafikus	16	Közepes	320 × 200	EGA
14	Grafikus	16	Nagy	640 × 200	EGA
15	Grafikus	4	Különösen nagy	640 × 350	EGA

Jó tudni, hogy a fekete-fehér és a színelnyomásos módok (a 0-s, a 2-es és az 5-ös mód) csak a kompozit kimeneten nyomják el a színt, az RGB-kimeneten nem.

A ROM-BIOS a mód beállításakor általában akkor is üríti a képernyőhöz tartozó puffertárat, ha ismételten azonos módot állítunk be. Ez egyben hatékony módszert is ad a képernyő törlésére. A Compaq-PC-vel kompatíbilis gépeknél ez a fajta képernyőtörlés nem ideális, mivel jelentős késleltetést okoz.

(☛ A megjelenítési módokról részletesebb adatokat olvasatunk a 4. fejezetben.) A 449-es tárrekesz leírásánál található a megjelenítési mód tárbeli feljegyzéséről információ. A 15-ös (hexa F) módnál láthatjuk, hogyan olvasható ki az aktuális megjelenítési mód.

Az 1-es szolgáltatás: a kurzor méretének beállítása

Az 1-es mód szabja meg a szöveges módokban a villogó kurzor alakját és méretét. Az IBM szabványos kurzora rendszerint 1 vagy 2 villogó sor a karakterpozíció alján. A kurzor méretét a megjelenített sorok számának újradefiniálásával változtathatjuk meg.

A színes-grafikus adapter 8 soros kurzort tud megjeleníteni. Ezeket felülről lefelé 0-tól 7-ig számozzuk. A monokrom adapter és az EGA kurzora 14 soros; ennek sorait szintén fentről lefelé, 0-tól 13-ig számozzuk. A kurzor méretét az első és az utolsó sor megadásával határozzuk

meg. (Ezek megegyeznek a BASIC **LOCATE**-utasításának "start" és "stop" paraméterével.) A kezdősor számát a *CH*-regiszterbe, a zárósorét pedig a *CL*-regiszterbe kell beírni. Alaphelyzetben $CH = 6$, $CL = 7$ a színes-grafikus adapternél, ill. $CH = 12$, $CL = 13$ a monokrom adapternél.

Ha a kezdősor száma kisebb a zárósor számánál, akkor egybefüggő kurzor jelenik meg. Ha a kezdősor száma nagyobb a zárósor számánál, akkor a kurzor két részből áll.

9.3. táblázat
A kurzor méretének beállításakor használt regiszterek az 1-es szolgáltatásnál

A szolgáltatások száma	Paraméterek	
AH=1	CH:	a kurzor kezdő sora
	CL:	a kurzor utolsó sora

A sorszámok a regiszterekben csak 3 bitet (0., 1., 2.) foglalnak el. Ha a *CH* 5. bitjét 1-re állítjuk (ez 32-nek felel meg), akkor a kurzor eltűnik. Ha grafikus mód van beállítva, akkor az 5. bit automatikusan 1 lesz, hogy megakadályozza a kurzor és a grafikus megjelenítés kölcsönös zavarását. Ez szöveges módban a kurzor eltüntetésének egyik lehetősége. Egy másik szerint a képernyőn kívülre visszük a kurzort, pl. a 26. sor 1. oszlopába. Mivel grafikus módban nincs valódi kurzor, az esetleg látható kurzort vagy a **CHR$(223)**-as kódú tömör-téglalap karakterrel, vagy a háttérattribútumok megváltoztatásával szimulálják.

(☛ A kurzorról további adatok találhatók a 4. fejezetben.) A 3-as szolgáltatás az iménti fordítottja: a kurzor pozíciójának leolvasása.

A 2-es szolgáltatás: a kurzor pozíciójának beállítása

A 2-es szolgáltatás a sor- és az oszlopkoordinátával állítja be a kurzor pozícióját. Szöveges módoknál több képernyőoldal is lehet, és mindegyikhez külön kurzorpozíció tartozik. Bár a grafikus módokban nincs látható kurzor, a szöveges módokhoz hasonlóan nyilvántartják a logikai kurzor pozícióját. Ezzel a logikai kurzorpozícióval lehet nyomon kísérni a karakteres I/O műveleteket.

9.4. táblázat
A kurzor pozíciójának beállítására használt regiszterek a 2-es szolgáltatásnál

A szolgáltatások száma	Paraméterek	
AH=2	DH:	sorszám
	DL:	oszlopszám
	BH:	oldalszám (grafikus módban 0-ra kell állítani)

A kurzor pozícióját úgy adjuk meg, hogy a *DH*-regiszterbe beírjuk a sor számát, a *DL*-be az oszlop számát, a *BH*-ba pedig az oldal számát. A sorok és az oszlopok számozása a bal felső sarokban a 0,0 koordinátájú pontban kezdődik. A grafikus módok is karakteres sor- és oszlopszámokkal tartják nyilván a kurzor helyzetét, nem a képpont koordinátáival. Az oldalak számozása a BASIC által is használt hagyományos számozást követi: a 40 oszlopos képernyőnél 0-tól 7-ig, a 80 oszloposnál 0-tól 3-ig. Grafikus módokban az oldal számát 0-ra kell beállítani.

(☞ A 9.4. táblázatban látható a regiszterek beállítása. A képernyőoldalakról további részleteket olvashatunk a 4. fejezetben. Ott található a szöveges megjelenítési formátumok leírása is.) A 3-as szolgáltatás (a kurzor pozíciójának leolvasása) az iménti szolgáltatás fordítottja.

A 3-as szolgáltatás: a kurzor pozíciójának kiolvasása

A 3-as szolgáltatás az 1-es és a 2-es fordítottja. Ha *BH*-ban adjuk meg az oldal számát, akkor a ROM-BIOS *CH*-ba írja a kurzor kezdősorát, *CL*-be pedig a zárósorát. A kurzor pozícióját *DH*-ban (sorszám) és *DL*-ben (oszlopszám) adja meg. Akárcsak a 2-es szolgáltatásnál, grafikus módban itt is 0-t kell megadni oldalszámként.

9.5. táblázat
A kurzor pozíciójának kiolvasására használt regiszterek a 3-as szolgáltatásnál

A szolgáltatások száma	Paraméterek
AH = 3	BH: oldalszám (grafikus módban 0-ra kell állítani)
	DH: sorszám
	DL: oszlopszám
	CH: a kurzor kezdő sora
	CL: a kurzor utolsó sora

(☞ A 9.5. táblázatban látható a regiszterek beállításának összefoglalása. A 4. fejezetben további részletek találhatók a képernyőoldalakról és részletesen leírjuk a szöveges megjelenítési formátumokat.)

A 4-es szolgáltatás: a fényceruza pozíciójának kiolvasása

A 4-es szolgáltatás a fényceruza állapotát jelzi: azt, hogy megnyomták-e, és ha igen, akkor hol van a képernyőn. Ha *AH* = 1, akkor megnyomták, ha *AH* = 0, akkor nem. A hardver értékeli, hogy a fényceruza melyik képpont előtt áll, ezt a ROM-BIOS kétféle formában is közli: *DH*-ban és *DL*-ben a karakter sor- és oszlopszámát; *CH*-ban és *BX*-ben pedig a képpont sor- és oszlopszámát. Mivel a képpont oszlopszáma 255-nél nagyobb is lehet, ezért egy teljes szóban kapjuk az eredményt. A többi értéket bájtban kell kezelni.

9.6. táblázat
A fényceruza pozíciójának kiolvasására használt regiszterek a 4-es szolgáltatásnál

A szolgáltatások száma	Paraméterek
AH=4	DH: a karakterek sorszáma
	DL: a karakter oszlopszáma
	CH: a képpont sorszáma (0–199)
	BX: a képpont oszlopszáma

Az 5-ös szolgáltatás: az aktív képernyőoldal beállítása

Az 5-ös szolgáltatás a 0-s, 2-es és 3-as szöveges megjelenítési módoknál beállítja az aktív képernyőoldalt. Az *AL*-regiszterben kell megadni az oldal számát. A 40 oszlopos módokban ez az érték 0 és 7 között, a 80 oszlopos módokban pedig 0 és 3 között lehet. Alapállapotba a 0-s lapot használjuk. A 0-s oldal a képernyőtár elején található, ezt követik 2K-nként (40 oszlopos módban) vagy 4K-nként (80 oszlopos módban) a további oldalak. A magasabb számú oldalak a tár magasabb címein találhatók.

(☛ A 4. fejezetben további részletek vannak a képernyőoldalakról.)

9.7. táblázat
Az aktív képernyőolal beállítására használt regiszter az 5-ös szolgáltatásnál

A szolgáltatások száma	Paraméterek
AH=5	AL: az új képernyőolal száma (0–3 a 2-es és 3-as módban, 0–7 a 0-s és 1-es módban)

A 6-os szolgáltatás: az ablak görgetése felfelé

A 6-os és a vele rokon 7-es szolgáltatás segítségével a képernyőn téglalap alakú szöveges ablakot definiálhatunk, és azt egy vagy több sorral felfelé vagy lefelé lehet görgetni. A gördülőhatás úgy jön létre, hogy a 6-os szolgáltatásnál alulról üres sorok lépnek be a képernyőre (7-es szolgáltatásnál a képernyő tetején), és az ablak felső sora (a 7-es szolgáltatásnál az alsó) kilép és eltűnik.

A görgetendő sorok számát az *AL*-ben kell megadni. Ha $AL=0$, akkor a teljes ablak törlődik (ugyanez történne, ha több sort akarnánk görgetni, mint amennyi az ablakban van).

9.8. táblázat
A görgetésnél használt ablak méretét megadó regiszterek a 6-os és a 7-es szolgáltatásnál

A szolgál-tatások száma	Paraméterek
AH = 6	AL: a görgetendő sorok száma
	CH: a bal felső sarok sorszáma
	CL: a bal felső sarok oszlopszáma
	DH: a bal alsó sarok sorszáma
	DL: a bal alsó sarok oszlopszáma
	BH: az üres sorok attribútuma

Az ablak helyét vagy méretét a *CX*- és a *DX*-regiszterben adjuk meg. A *CH* tartalmazza a felső sort, a *DH* az alsót; a *CL* a bal oldali oszlopot, a *DL* a jobb oldalit. A belépő üres sorok megjelenítési attribútumát *BH*-ból veszi. A 9.8. táblázatban látható 6-os és 7-es szolgáltatás regiszterének beállítását foglalja össze.

Az ablak görgetése rendszerint kétlépéses: ha van egy új sor, amit az ablakba ki kell írni, a 6-os vagy a 7-es szolgáltatás segítségével az ablak tartalmát görgetjük. Ezután következik az új sor kiírása a kurzorbeállító és a karakterkiíró szolgáltatással. A következő példa ezt az ablakkezelést szemlélteti.

```
DEBUG          ; a DEBUG behívása
A              ; az utasítások fordításának kezdeményezése
INT 10         ; a hexa 10-es megszakítást kérő utasítás
[Return]       ; a fordítás befejezése
R AX           ; az AX tartalmának olvasása és megváltoztatása
0603           ; a 6-os szolgáltatás, 3 soros ablakkal
R CX           ; a CX tartalmának kiírása és megváltoztatása
050A           ; a bal felső sarok meghatározása: 5. sor, 10. oszlop
R DX           ; a DX tartalmának kiírása és megváltoztatása
1020           ; a jobb alsó sarok meghatározása: 16. sor, 32. oszlop
D 0 L 180      ; a képernyő feltöltése értelmetlen karakterekkel
G = 100 102    ; az INT 10 végrehajtása, majd megállás
```

(☛ A 8. fejezetben részletesen foglalkozunk az assembly nyelvű szubrutinokkal. A DEBUG leírását az IBM DOS kézikönyvben találhatjuk meg.)

A 7-es szolgáltatás: ablak görgetése lefelé

A 7-es szolgáltatás a 6-os tükörképe. A két szolgáltatás közötti lényeges különbség a görgetési művelet iránya. A 7-es szolgáltatásnál az üres sorok az ablak tetején jelennek meg, a régiek pedig az alján tűnnek el. A 6-os szolgáltatásnál fordított művelet játszódik le. A paraméterek beállításának leírása a 6-os szolgáltatásnál található.

A 8-as szolgáltatás: karakter és attribútumának kiolvasása

A 8-as szolgáltatással a karakterek egyenesen a képernyőről olvashatók le, vagyis közvetlenül a képernyőtárból. Ez a szolgáltatás különösen előnyös, mert szöveges és grafikus módban egyaránt működik.

Grafikus módban a szolgáltatás a karakterek kirajzolására szolgáló táblázattal és mintaillesztéssel ismeri fel a karaktert. Ez a szolgáltatás még saját tervezésű karaktereinket is képes felismerni. Szöveges módban természetesen az ASCII-kód közvetlenül megtalálható a képernyőtárban.

A 8-as szolgáltatás az *AL*-regiszterben adja meg a képernyőről leolvasott karakter ASCII kódját. Ha grafikus módban a karakter egyetlen szabványos ASCII-kóddal sem egyezik meg, akkor az *AL*-regiszterben 0-t kapunk. Szöveges módban a szolgáltatás a színattribútumokat is megadja az *AH*-regiszterben, a *BH*-ban pedig a képernyőoldal számát. Grafikus módba viszont nem adja meg az oldal számát.

9.9. táblázat
Szövegkarakter és attribútum kiolvasására használt regiszterek a 8-as szolgáltatásnál

A szolgáltatások száma	Paraméterek
AH=8	BH: az aktív képernyőoldal (grafikus módoknál szükségtelen)
	AL: a kurzor pozíciójából kiolvasott karakter ASCII kódja
	AH: a szöveg karakterattribútuma

(☛A 4. fejezetben írtuk le részletesen a szöveges karaktereket és az attribútumbájtokat. Ott találhatók további részletek a szöveges és grafikus módú karakterekről. Az ASCII-karakterek a *C* függelékben találhatók.)

A 9-es szolgáltatás: karakter és attribútum kiírása

A 9-es szolgáltatás egy karaktert és a hozzá tartozó színattribútumot írja ki egy vagy több példányban. A karaktert az *AL*-regiszterben kell megadni, a szöveges módú attribútumot és a grafikus mód színét pedig *BL*-ben. A *CX*-regiszterben kell megadni, hogy a karaktert hány példányban kell kiírni.

Szöveges módban a *BH*-ban kell megadni a képernyőoldal számát, grafikus módban ez nem szükséges.

A karakter és attribútumának megjelenítése az előírt számba a kurzor jelenlegi pozíciójában kezdődik. Bár a kurzor helyben marad, a karakterek egymás utáni helyeken jelennek meg. Szöveges módban egy sor betelte után a kiírás új sorban folytatódik, ez tovább növeli e szolgáltatás használhatóságát. Grafikus módban a karakterek egy sor betelte után nem folytatódnak új sorban.

A 9-es szolgáltatást egyformán használhatjuk egyes karakterek, ill. egy-egy karakter ismételt kiírására. A karakter ismételt kiírását szóközök vagy más karakterek gyors egymás

9.10. táblázat
Szövegkarakter és attribútum írására használt regiszterek a 9-es szolgáltatásnál

A szolgáltatások száma	Paraméterek
AH=9	AL: a kiírandó ASCII-karakter BL: a kiírandó karakterattribútuma BH: aktív képernyőoldal (grafikus módoknál szükségtelen) CX: a karaktert és attribútumát ennyiszer kell kiírni

mellé írásra használják: pl. egy téglalap vízszintes oldalának kirajzolására. (☛ L. a *C* függeléket.) Ha a karaktert egy példányban akarjuk kiírni, akkor győződjünk meg arról, hogy a *CX*-ben 1 van-e. Ha 0-ra van állítva, akkor az ismétlés világgá szalad.

A 9-es szolgáltatás előnye a hasonló jellegű 14-essel szemben, hogy szabályozni tudjuk a színeket. Ugyanakkor megvan az a hátránya, hogy a kurzor nem mozog előre automatikusan.

Grafikus módban a *BL*-regiszterben megadott szín az előtér színe, ilyen színű képpontokból áll össze a karakter. Ha a 7-es bit értéke 1 (ez a 128 vagyis a hexa 80 értéknek felel meg), akkor a *BL*-ben megadott színbitek kizáró VAGY-művelettel kombinálódnak a képpont aktuális színével. Ezzel a módszerrel könnyen biztosíthatjuk, hogy az eredményül kapott szín eltér az eredeti színtől, így szinte teljes biztonságot ad az olvashatóságra. Ha a *BL* 7-es bitje 0, akkor a *BL*-ben megadott szín egyszerűen felülírja a képpont előző színét. Ugyanez érvényes a karaktereket és képpontokat író (hasonló jellegű) 10-es és 12-es szolgáltatásra.

(☛ A szöveges módú attribútumokról és a grafikus mód színattribútumairól további részletek találhatók a 4. fejezetben.)

A 10-es szolgáltatás: (hexa A) karakterkiírás

A 10-es szolgáltatás – egy különbségtől eltekintve – megegyezik a 9-es szolgáltatással. A 9-es szolgáltatás lehetővé tette szöveges módban a képernyő színattribútumának megváltoztatását, a 10-es nem. Ennek ellenére graikus módban mégis meg kell adni a színt *BL*-ben, ez azonban

9.11. táblázat
Egy karakter kiírására használt regiszterek a 10-es szolgáltatásnál

A szolgáltatások száma	Paraméterek
AH=10	AL: a kiírandó ASCII-karakter BL: a színattribútum grafikus módokban BH: az aktív képernyőoldal CX: ennyiszer kell a karaktert kiírni

egy karakterkiíró szolgáltatás, könnyen összezavarhatja a specifikációt. A színre ugyanaz érvényes, mint a 9-es és 12-es szolgáltatásnál: a szín használható közvetlen módon, de létre jöhet az ott levő színnel végzett kizáró VAGY-művelet eredményeként is. (A magyarázatot l. a 9-es szolgáltatásnál.)

☛ A 4. fejezetben a szöveges mód attribútumairól, és a grafikus mód színattribútumairól találhatunk további részleteket.

A 11-es szolgáltatás (hexa B): a színpaletta beállítása

A 11-es szolgáltatással a közepes felbontású grafika két palettája közül lehet választani. Ezt úgy érjük el, hogy a *BH*-ba töltjük a paletta színazonosítóját, a *BL*-be pedig a szín értékét. (☛ A 4. fejezetben találhatunk további részleteket a színpalettákról.)

A szolgáltatás egyik változata szöveges módban, a többi a grafikus módban érvényes. Ha szöveges módban *BH* értéke 0, akkor *BL* adja meg a szöveg területét körülvevő sáv színét. Ezt a színt a teljes 16 színű palettából lehet kiválasztani. Ha grafikus módban *BH* értéke 0, akkor *BL* tartalmazza a háttér és a keret színét is. A keret egybemosódik a képernyő azon területeivel, amelyeknek színe a háttér színével egyezik meg. A *BL* értékét a teljes 16 színű palettáról lehet kiválasztani.

Ha *BH* értéke 1, akkor *BL* a használt palettát választja ki. A színes-grafikus adapternél ez csak a 4-es módra vonatkozik: közepes felbontás, négyszínű grafika. Fejlettebb képernyő-adapterek esetében (ilyen pl. a PCjr adaptere) más módokra is vonatkozhat. Most csak a 4-es módnál is megengedett szabványos négyszínű palettával foglalkozunk: ezek a 0-s és az 1-es paletták. A paletta számát a *BL* értéke választja ki.

A 0-s paletta 4 színe a következő:

- 0: a jelenlegi háttérszín;
- 1: zöld (2);
- 2: vörös (4);
- 3: barna (6).

Az 1-es paletta 4 színe a következő:

- 0: a jelenlegi háttérszín;
- 1: ciánkék (3);
- 2: bíborvörös (5);
- 3: fehér (7).

9.12. táblázat
A színpaletta beállítására használt regiszterek a 11-es szolgáltatásnál

A szolgál-tatások száma	Paraméterek
AH = 11	BH: a szín azonosítója (0 vagy 1, 320 × 200 pontos grafikánál) BL: a szín azonosítóhoz rendelendő szín- vagy palettaérték

A 12-es szolgáltatás (hexa C): a képpont kigyújtása

A 12-es szolgáltatás egyetlen képpontot gyújt ki. Mivel a 9-es, 10-es és 14-es szolgáltatásnál használt kurzor karakterpozíciókra vonatkozik, ennél a szolgáltatásnál meg kell adni raszterben a sor és a képpont oszlopszámát. A koordináták számozása szokás szerint 0,0-tól kezdődik, ez a képernyő bal felső sarka.

9.13. táblázat
Egy képpont kigyújtására használt regiszterek a 12-es szolgáltatásnál

A szolgáltatások száma	Paraméterek
AH = 12	AL: a képpont színkódja (0 – 15)
	DL: a képpont sorszáma
	CX: a képpont oszlopszáma

A sorszám megadásához 1 bájt elég, erre a *DL*-t használjuk. Az oszlop megadására 1 bájt kevés, így erre a *CX*-et használjuk. A színt *AL*-ben kell megadni, itt is létrehozható közvetlenül vagy kizáró VAGY-művelettel. (☛ Ennek magyarázatát l. a 9-es szolgáltatásnál).

(☛ A 4. fejezetben grafikus módokban ismertetjük a képpontokat.)

A 13-as szolgáltatás (hexa D): a képpont kiolvasása

A 13-as szolgáltatás a 12-es fordítottja: a képpont kigyújtása helyett kiolvas egy fénypontot. A képpontnak mindössze egyetlen színattribútuma van, éppen ezt az információt adja meg a 13-as szolgáltatás. (A 8-as szolgáltatás a színt és az ASCII-kódot is megadja.) Nem *DX*-ben, hanem *DL*-ben adjuk meg a sort (☛ l. a megjegyzést a 12-es szolgáltatásnál), az oszlopot pedig *CX*-ben. A képpont színkódját *AL*-ben kapjuk. Várakozásunkhoz híven a magas helyiértékű bitek értéke 0 lesz.

9.14. táblázat
Egy képpont kiolvasására használt regiszterek a 13-as szolgáltatásnál

A szolgáltatások száma	Paraméterek
AH = 13	AL: a képpont színkódja (0 – 15)
	DL: a képpont sorszáma
	CX: a képpont oszlopszáma

A 14-es szolgáltatás (hexa E): karakterkiírás teletype módban

A 14-es szolgáltatást használjuk leggyakrabban karakterkiírásra. A karaktereket egyesével írja a képernyőre TTY (teletype) módban. Ilyenkor a képernyő úgy viselkedik, mint a legegyszerűbb, legnyersebb nyomtató –, nekünk éppen erre van szükség a szokványos karakterkiírásnál. Nem lehet tekintettel olyan finomságokra, mint a szín, a villogó karakter vagy a kurzor pozíciójának vezérelhetősége.

Ha ezt a szolgáltatást használjuk, akkor a karakter a kurzor pillanatnyi helyére íródik ki, a kurzor egy pozícióval előrehalad, közben szükség szerint sort vált, vagy felfelé görgeti a képernyőt. A kiírandó karaktert *AL*-ben kell megadni.

Szöveges módban a jelenlegi képernyő-attribútumok karakterről karakterre megmaradnak. Grafikus módban az előtér színét mindig meg kell adni *BL*-ben is.

9.15. táblázat
Teletype-módban kiírt karakterhez használt regiszterek a 14-es szolgáltatásnál

A szolgál-tatások száma	Paraméterek
AH = 14	AL: a kiírandó ASCII-karakter
	BL: a karakter előtérszíne (csak grafikus módban)
	BH: az aktív képernyőoldal (grafikus módokban nem szükséges)

Van négy karakter, amelyre a 14-es szolgáltatás az ASCII jelentése szerint reagál: a **CHR$(7)** dudál, a **CHR$(8)** visszalép, a **CHR$(10)** sort emel, a **CHR$(13)** pedig a sor elejére áll. Az összes többi karaktert egyszerűen kiírja.

A 9-es szolgáltatással szemben a 14-es legfőbb előnye, hogy a kurzor automatikusan mozog. A 9-es szolgáltatással viszont megadhatjuk a színattribútumokat is. De jó lenne a kettőt egyesíteni!

9.16. táblázat
A megjelenítési mód kiolvasására használt regiszterek a 15-ös szolgáltatásnál

A szolgál-tatások száma	Paraméterek
AH = 15	AL: az aktuális megjelenítési mód
	AH: a karakterek száma soronként
	BH: az aktív képernyőoldal (grafikus módoknál 0)

A 15-ös szolgáltatás (hexa F): az aktuális megjelenítési mód kiolvasása

A 15-ös szolgáltatás megadja az aktuális megjelenítési módot. Közöl még két hasznos adatot is: a sor szélességét (20, 40 vagy 80 karakter) és a képernyőoldal számát.

A megjelenítési módot (a 0-s szolgáltatásnál már elmondtuk) *AL*-ben kapjuk meg, a sor hosszát pedig *AH*-ban. Ez azt tartalmazza, hogy egy sorba hány karakter fér. (A durva felbontású grafikánál a helyes 20-as értéket adja meg.) A képernyőoldal számát *BH*-ban kapjuk meg. Grafikus módoknál az oldal számát nullára kell beállítani. A 9.16. táblázatban foglaltuk össze a regiszterek tartalmát.

(☛ A 4. fejezetben találhatók a megjelenítési módok részletei. A 3. fejezetben (a hexa 449-es tárcímnél) pedig a megjelenítési mód tárolásának leírása.)

A 19-es szolgáltatás (hexa 13): a karakterlánc kiírása

AT 

A 19-es szolgáltatást csak az AT nyújtja. Ez lehetővé teszi egy karakterlánc kiírását a képernyőre. A szolgáltatást alkotó 4 alárendelt szolgáltatás segítségével a karakterek attribútumait egyesével vagy csoportosan is megadhatjuk. Az alárendelt funkciótól függően a kurzor a karakterlánc végére kerül, vagy helyben marad.

Az alárendelt szolgáltatás számát *AL*-be, a lánc elejére irányított mutatót *ES* : *BP*-be, a lánc hosszát pedig *CX*-be kell tenni. A lánc kezdő pozícióját a képernyőn *DX*-be, a képernyőoldal számát *BH*-ba kell betölteni.

A 0-s és 1-es alárendelt szolgáltatás a *BL*-regiszterben megadott attribútummal írja ki a karakterláncot. A 0-s szolgáltatásnál a kurzor nem mozdul a *DX*-ben megadott pozícióról, az 1-es szolgáltatásnál pedig az utolsó kiírt karakter mögé kerül.

A 2-es és 3-as alárendelt szolgáltatás karaktereket és attribútumokat ír a képernyőre. Először kiírják a karaktert, majd az attribútumot. A 2-es szolgáltatásnál a kurzor a *DX* értékével megadott helyen marad, a 3-asnál az utolsónak kiírt karakter mögé kerül.

Megjegyzés és példa

A ROM-BIOS videoszolgáltatásai közt barangolva áttekintettük az egyes szolgáltatások működését. Ezen ismeretek birtokában a következő kérdés szokott felmerülni: válasszuk-e a ROM-BIOS-szolgáltatások közvetlen használatát vagy a DOS-t, esetleg egy másik programozási nyelv magasabb szintű szolgáltatásait. Erre általában azt válaszolhatjuk: mindig a legmagasabb szintűt kell választani, hogy a célt elérjük. Ebben az esetben nem szól semmi nyomós indok a ROM-BIOS-szolgáltatások használata ellen, nem okozhatunk velük nagyobb kárt. A következő fejezetben azonban a lemezes szolgáltatásoknál mást tanácsolunk majd, javasolva a ROM-BIOS-szolgáltatások mellőzését, mert használatuk meglehetősen kockázatos.

A PC-modellek videolehetőségei jelentősek, a ROM-BIOS lehetővé teszi ezek teljes kihasználását. A 14 – 18. fejezetben látni fogjuk, hogy a DOS szolgáltatásai szegényesek, csak a legegyszerűbb karakteres szolgáltatásokat biztosítják. Hasonlóan sok programozási nyelv is (pl. a Pascal és a C) csak a DOS-szolgáltatások kissé javított változatát biztosítja – és semmi többet. Ha tehát szükség van a PC kifinomultabb videolehetőségeire, és nem a BASIC-et használjuk (amely ezeket a szolgáltatásokat biztosítaná), akkor igénybe kell venni a ROM-

BIOS szolgáltatásait. A ROM-BIOS használatát a leginkább az indokolja, hogy képes a képernyő fölötti teljes ellenőrzésre.

A ROM-BIOS szolgáltatásainak közvetlen használata rendszerint szükségessé tesz egy assembly nyelvű interfészt. Ezért példánkban egy Pascalban használható formájú modult ismertetünk. A modul beállítja az 1-es megjelenítési módot (40 oszlopos, szöveges színes); és a háttér színét kékre állítja.

Ime az assembly modul! (☛ A formátummal kapcsolatban l. a 8. fejezetet):

```
MODULE    SEGMENT    'CODE'
          PUBLIC     BLUE40
BLUE40    PROC       FAR
          PUSH       BP          ; a régi bázismutató mentése
          MOV        BP,SP       ; az új bázismutató beállítása
; a megjelenítési mód beállítása
          MOV        AH,0        ; a 0-s szolgáltatás: módbeállítás
          MOV        AL,1        ; 1-es mód: színes 40 oszlopos szöveg
          INT        16          ; a videoszolgáltatás kérése
; a háttérszín beállítása
          MOV        AH, 11      ; 11-es szolgáltatás: színbeállítás
          MOV        BH,0        ; a háttérbeállítás
          MOV        BL,1        ; az 1-es szín használata: kék
          INT        16          ; a videoszolgáltatás kérése
          POP        BP          ; a régi bázismutató visszaállítása
          RET        0           ; a visszatérés a hívóhoz
BLUE40    ENDP
MODULE    ENDS
          END
```

10 A ROM-BIOS lemezes szolgáltatásai

Bevezetés • 162

A szabványos ROM-BIOS lemezes szolgáltatásai • 162

A 0-ás szolgáltatás: a lemezes rendszer alapállapotba hozása • 163
Az 1-es szolgáltatás: a lemezstátusz beolvasása • 163
A 2-es szolgáltatás: szektorok beolvasása • 164
A 3-as szolgáltatás: szektorok felírása • 165
A 4-es szolgáltatás: szektor verifikálása • 165
Az 5-ös szolgáltatás: sáv formázása • 165

Az AT merevlemezes szolgáltatásai • 167

A 8-as szolgáltatás: a meghajtó aktuális paramétereinek olvasása • 167
A 9-es szolgáltatás: a merevlemez paramétertáblázatainak inicializálása • 167
A 10-s és 11-es szolgáltatás (hexa A és B): hosszú szektorok felírása és leolvasása • 168
A 12-es szolgáltatás (hexa C): a cilinder megkeresése • 168
A 13-as szolgáltatás (hexa D): a lemez alapállapotba hozása (alternatív lehetőség) • 168
A 16-os szolgáltatás (hexa 10): a meghajtó készenléti állapotának ellenőrzése • 168
A 17-es szolgáltatás (hexa 11): a meghajtó újrakalibrálása • 168
A 20-as szolgáltatás (hexa 14): a meghajtó diagnosztikája • 168
A 21-es szolgáltatás (hexa 15): a lemez típusának leolvasása • 168
A 22-es szolgáltatás (hexa 16): a lemez státuszának megváltoztatása • 169
A 23-as szolgáltatás (hexa 17): a lemez típusának beállítása • 169

A lemez alaptáblázata • 169

Megjegyzések és példák • 171

Bevezetés

Most áttekintjük a ROM-BIOS lemezes szolgáltatásait. (☛ A lemez logikai szerkezetének jobb megértéséhez célszerű átnézni az 5. fejezetet. A DOS által biztosított magasabb szintű szolgáltatásokat a 15 – 18. fejezetben találhatjuk.)

Általában a lemezes műveleteket legjobb a lemezes operációs rendszerre bízni. Aki mégis úgy dönt, hogy használni fogja valamelyik ROM-BIOS-szolgáltatást annak célszerű előbb elolvasni a fejezet Megjegyzések és példák című részét.

A szabványos ROM-BIOS lemezes szolgáltatásai

Mivel a hajlékonylemez meghajtója csak néhány egyszerű műveletet tud elvégezni, csak hat szabványos BIOS-lemezszolgáltatás áll rendelkezésre az összes PC-modellen. [AT] Mivel az AT jóval bonyolultabb lemezegységgel van fölszerelve, sok új szolgáltatással bővítette a ROM-BIOS szolgáltatásait. A bővítéseket külön ismertetjük.

A ROM-BIOS lemezes szolgáltatásait a 19-es megszakításon keresztül vehetjük igénybe; a szolgáltatást azonosító számot az *AH*-regiszterbe kell beírni. A hat szabványos szolgáltatást szokás szerint 0-tól 5-ig számozzuk. (L. 10.1. táblázatot.)

10.1. táblázat
A ROM-BIOS 6 szabványos lemezes szolgáltatása

Szolgáltatás	Feladata
0	A lemezes rendszer alapállapotba hozása
1	A lemezstátusz beolvasása
2	A lemezszektorok leolvasása
3	A lemezszektorok felírása
4	A lemezszektorok verifikálása
5	A lemezsáv formázása

A lemezes szolgáltatások a lemez alaptáblázatának felügyelete alatt állnak. Ez a táblázat több, mint egy tucat olyan paramétert tárol ROM-ban, ami a lemez működtetéséhez szükséges (pl. a szektor méretét, a léptetés sebességét, a fej beállási idejét). A legtöbb programozó számára a lemez alaptáblázata a lemezes szolgáltatások észrevehetetlen része. Alkalmanként azonban előfordulhat, hogy egyes paramétereit meg kell változtatni. Ezért a fejezet végén röviden ismertetjük a táblázatot.

JR 

📖 MEGJEGYZÉS: Bár a PCjr ROM-BIOS lemezes szolgáltatásai megegyeznek az IBM többi személyi számítógépével, a PCjr lemezmeghajtóinak működése alapvetően különbözik azokétól. Ennek elsődlegesen az az oka, hogy a PCjr hajlékonylemezes meghajtója nem használja a közvetlen memória-hozzáférést. Az adatmozgatást a ROM-BIOS-szoftver végzi, és ez lemezes műveletek közben leköti a PCjr számolási

kapacitását. Ez viszont a lemezes szolgáltatások időviszonyaiban és teljesítményében határozott különbségként mutatkozik meg. Ennek többek között az is következménye, hogy néhány olyan másolás elleni védelmi rendszer, amit más modellre terveztek, a PCjr-en nem működik.

A 0-s szolgáltatás: a lemezes rendszer alapállapotba hozása

A 0-s szolgáltatás a hajlékonylemezes vezérlőt és a meghajtót alapállapotba hozza. A szolgáltatás nincs hatással a meghajtóban esetleg benn lévő lemezekre. A 0-s szolgáltatás hatására a ROM-BIOS kiszolgáló szubrutinjai tiszta lappal indulnak a következő lemezes műveletnél. Ez magába foglalja a lemezek író – olvasó fejének újrakalibrálását. Ez a művelet a fejet egy bizonyos sáv fölé pozicionálja. Programjainkban ezt a szolgáltatást általában a lemezmeghajtóban előforduló hiba után szoktuk használni.

Az 1-es szolgáltatás: a lemezstátusz beolvasása

Az egyes szolgáltatás a hajlékonylemez státuszát közli az *AL*-regiszter 8 bitjén. A státusz minden lemezes művelet után megmarad, a később ismertetendő olvasó, író, verifikáló és formázó művelet után is. A státusz megőrzése lehetővé teszi, hogy egy hibakezelő vagy hibaelemző szubrutin a lemezt működtető szubrutinoktól teljesen függetlenül működjön. Ez esetenként nagyon hasznos lehet. Megfelelő körülmények között a DOS-ra vagy egy programozási nyelvre bízhatjuk a lemez kezelését, és ez nagyon bölcs döntés. (☛ L. a Megjegyzések és példák részt a fejezet végén.) Ugyanakkor saját programokkal derítjük ki és közöljük a hibajelenség finomabb részleteit. (A 10.2. táblázatban láthatjuk a státuszbájtot részletesen.)

10.2. táblázat
Az 1-es szolgáltatás ebben a formában adja meg AL-ben a lemez státuszbájtját

Bit 7 6 5 4 3 2 1 0	Értéke (Decimális)	Jelentése
1	128	Időtúllépés: a meghajtó nem válaszolt
. 1	64	Hibás keresés: nem sikerült az előírt sávra ráállni
. . 1	32	Vezérlőhiba: a lemezvezérlő működésében van hiba
. . . 1	16	CRC-hiba: olvasáskor a paritásösszeg nem egyezett
. . . . 1 . . .	8	DMA-hiba (PCjr-re nem vonatkozik)
. 1 . .	4	Hibás szektor: a kért szektor nincs a lemezen
. 1 .	2	Hibás címjel: a szektor azonosítójele hibás vagy hiányzik
. 1	1	Hibás parancs: a lemezvezérlő értelmetlen parancsot kapott
. . . . 1 . . 1	9	A DMA-határ túllépése: kísérlet 64 K-nál nagyobb terület elérésére (PCjr-nél nem érvényes)
. 1 1	3	Írásvédelem-hiba: kísérlet írásvédelemmel ellátott lemezre írásra

A 2-es szolgáltatás: szektorok beolvasása

A 2-es szolgáltatás egy vagy több lemezszektort olvas be a tárba. Ha több szektort akarunk beolvasni, akkor azoknak azonos sávon és oldalon kell lenniük. Ennek nagyrészt az az oka, hogy a ROM-BIOS nem tudja, hány szektor lehet egy sávon; tehát azt sem tudja, mikor kell egyik oldalról vagy sávról a másikra váltani. Ezt a szolgáltatást vagy egy-egy szektor beolvasására használjuk, vagy éppen ellenkezőleg, az egyik oldalon, egy teljes sáv szektorainak egyidejű beolvasására. Ez utóbbi fordul elő pl. a DOS DISKCOPY programjában, ahol nagy tömegű adattal kell dolgozni. Az olvasási műveletben több regiszter is tartalmaz vezérlő információkat (ezeket a 10.3. táblázatban foglaljuk össze):

10.3. táblázat
Az író, olvasó és verifikáló műveleteknél a vezérlő információt tartalmazó regiszterek

Paraméterek	Státuszeredmények
DL: meghajtószám	Ha CF = 0, akkor hibátlan, és AH = 0
DH: az oldalak vagy fejek száma egy lemezen	Ha CF = 1, akkor hiba van, és AH-ban vannak az 1-es szolgáltatás státuszbitjei
CH: a sáv száma	
CL: a szektor száma	
AL: a leolvasandó szektorok száma	
ES : BX: a puffer címe	

A *DL*-ben van a meghajtó száma.

A *DH*-ban a hajlékonylemez oldala, ill. a fejszáma (0 vagy 1).

A *CH*-ban van a sáv száma. Ez rendszerint 0 és 39 közötti érték, de bizonyos esetekben több is lehet (mint néhány másolás ellen védő rendszernél). Sok meghajtó 42 sávval is működik.

A *CL*-ben van a szektorszám. A szektorszám rendszerint 1 és 8 vagy 1 és 9 közti érték, de másolás ellen védő rendszereknél ennél nagyobb is lehet. Figyeljük meg, hogy a szektorok számozása (a fejtől és sávtól eltérően) 1-gyel kezdődik.

Az *AL*-ben van a leolvasandó szektorok száma. Ez rendszerint 1, 8 vagy 9. Az IBM külön figyelmeztet rá, hogy a 0 értéket kerüljük.

Az *ES : BX*-ben van a puffer címe. Az adatokat befogadó tárterület címét szegmentált cím formájában tartalmazza ez a regiszterpár. A szegmentált címek tárolására az összes ROM-BIOS-szolgáltatás általában az *ES : BX* regiszterpárt használja.

Az adatterületnek elég nagynak kell lennie a teljes beolvasott adatmennyiség tárolásához. A DOS-szektor szokásos mérete ugyan 512 bájt, de előfordulhatnak 1024 bájtos szektorok is. (☛ L. a következő formázó szolgáltatásokat). Ha a szolgáltatással egynél több szektort olvasunk be, akkor ezek a tárban egymás utáni területekre kerülnek.

A *CF* az eredmény sikerességének jelzője. A művelet sikerét valójában a *CF* és az *AH* együtt tartalmazza. Ha az átviteljelző *(CF)* értéke 0, akkor nem volt hiba, és az *AH* is 0. Ha a *CF* értéke 1, akkor az *AH* a hiba okát is tartamazza (az 1-es szolgáltatásnál részletezett formában).

Ha a 2-es (vagy bármelyik más) aktív szolgáltatást használjuk, akkor gondolni kell arra, hogy a motornak idő kell, míg a megfelelő sebességre felpörög. Ezt egyik szolgáltatás sem várja ki. Valójában ez ritkán jelent gondot, az IBM mégis azt javasolja, hogy a szolgáltatásokat használó programok csak három próbálkozás után gyanakodjanak valódi hibára, és a próbálkozások között hozzák a meghajtót alapállapotba. A javasolt műveleti sorrend (részben BASIC-ben) a következő:

```
10 HIBA.SZAM=0
20 WHILLE HIBA.SZAM<3
30 'hajtsd végre (olvasás, írás, verifikálás vagy formázó) művelet
40 'hiba ellenőrzés: ha nincs hiba, menj a 90-es sorra
50   HIBA.SZAM=HIBA.SZAM+1
60 'hajtsd végre alapállapotba hozó műveletet
70 WEND
80 'cselekedj, ahogy hiba esetén kell
90 'folytasd sikeres művelet után
```

(☛Nézzük meg, mit mond a fejezet alaptáblával foglalkozó része a lemez alaptáblázatának az alapállapotba hozó műveletre gyakorolt hatásáról!)

A 3-as szolgáltatás: szektorok felírása

A 3-as szolgáltatás egy vagy több szektort ír a lemezre. Ez a 2-es szolgáltatás fordítottja. A 2-es szolgáltatásnál elmondottak (regiszterek, részletek, megjegyzések) érvényesek a 3-as szolgáltatásnál is. (☛ L. még a 10.3. táblázatot.) A lemez szektorait felírás előtt meg kell formázni.

A 4-es szolgáltatás: szektor verifikálása

A 4-es szolgáltatás egy vagy több szektor tartalmát verifikálja. Nem arról van szó, amire sokan gondolnak: a művelet nem a lemezen tárolt adatokat hasonlítja össze a memóriában tároltakkal. A szolgáltatás által nyújtott verifikáció mindössze annyit jelent, hogy a szektorok megtalálhatók, leolvashatók, és a ciklikus redundanciát ellenőrző összeg (CRC, cyclical redundancy check) helyes. A CRC nagyon hatékony paritás-ellenőrző összegként működik, megbízhatóan kimutatja a szektorban található hibákat (pl. az elveszett vagy összekeveredett biteket).

A verifikációs műveletet leggyakrabban a 3-as számú szolgáltatás (lemezre írás) eredményének ellenőrzésére használjuk, de akármikor verifikálható a hajlékonylemez bármelyik része. Sokan a verifikációt szükségtelennek tartják a meghajtók megbízhatóságára és a hagyományos hibaészlelés hatékonyságára hivatkozva. A DOS nem verifikálja az írási műveletet, hacsak a **VERIFY ON** paranccsal külön nem kérjük rá.

A verifikációs művelet az író – olvasó műveletekhez hasonlóan működik, ugyanazokat a regisztereket is használja. Az egyetlen különbség az, hogy a verifikálás nem használt tárat, ezért az *ES : BX* regiszterpárra sincs szüksége.

Az 5-ös szolgáltatás: sáv formázása

Az 5-ös szolgáltatás a lemez egy oldalán egy sávot formáz. A formázó szolgáltatás az író – olvasó művelethez hasonlóan működik, de a *CL*-regiszterben tárolt szektorszámra nincs szükség. A 10.3. táblázatban látható többi paramétert a regiszterekben kell átadni, ill. az eredményt ott kapjuk meg.

Mivel a művelet egyszerre egy teljes sávot formáz, nincs mód egyetlen szektor formázására. Lehetséges azonban, hogy egy sáv szektoraihoz egyedi jellemzőket adjunk meg.

Minden szektornak 4 leíróbájtja van. Ezek az *ES* : *BX* által kijelölt tárterületen találhatók. Azok a címjelek, amelyekkel később az író, olvasó és verifikáló műveletek az egyes szektorokat azonosítják. A 4 címbájt neve: *C* (cilinder, ez a sávnál általánosabb fogalom), *H* (fej- vagy oldalszám), *R* (rekord- vagy szektorszám), és *N* (a szektorban található bájtok száma). Az utolsót méretkódnak is nevezik. Az *AL*-ben megadott szektorokhoz egy 4 bájtos mezőnek is tartoznia kell.

Egy szektor olvasásakor vagy írásakor a ROM-BIOS megkeresi a sávon a szektor azonosítóját. Ennek leglényegesebb része az *R*, vagyis a rekord-, ill. szektorszám. A cilinder és a fejszám nem igazán használt adat a címjelben, mivel a fej mechanikusan áll a megfelelő sávra, a fej kiválasztása pedig elektronikus, biztonsági okokból mégis rögzíti és ellenőrzi is.

A méretkód *(N)* értéke a 10.4. táblázatban látható szabványos értékek közül bármelyik lehet. A szokásos érték a 2-es, ez 512 bájtot jelent.

10.4. táblázat
Az N méretkód 4 szabványos értéke

N	Szektorméret, bájt	Szektorméret, K
0	128	1/8
1	256	1/4
2	512	1/2
3	1024	1

A szektorok lemezre írásának sorrendjét a címbájtok határozzák meg, a felírás nem feltétlenül szekvenciális. A valóságban a szektorok sorrendjét át lehet rendezni (ezt interleavenek hívják) a teljesítmény növelése érdekében, az időviszonyok megváltoztatása miatt vagy a másolás elleni védelem céljából. [XT] Az XT merevlemezén a szektorok sorrendje a következő: a logikailag szomszédos szektorok fizikailag 6 szektorral vannak eltolva. A DOS hajlékonylemezén a szektorok sorrendje szekvenciális: 1, 2, 3 stb.

A hagyományos 9 szektoros DOS hajlékonylemezén a 0-s sáv 1. oldalán a formátumcímek így néznek ki:

C H R N	*C H R N*	*C H R N*	...	*C H R N*
0 1 1 2	0 1 2 2	0 1 3 2	...	0 1 9 2

Formázás közben a meghajtó az indexlyukat figyeli, és ezt használja kiindulási jelként. Az indexlyukat a többi műveletnél figyelmen kívül hagyja, és a sávokat a címjelzésekkel kezeli.

Figyeljük meg, hogy a formázó szolgáltatás adatai között nem szerepel az, hogy milyen adattal kell feltölteni a megformázott szektorokat. Ez a lemez alaptáblázatából derül ki. (☛ L. később, a fejezet lemezalaptáblázattal foglalkozó részét.)

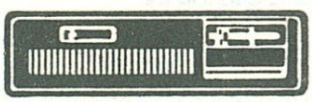
Az 5-ös szolgáltatás felhasználása másolás elleni védelemre

A sávokat rendkívül ravaszul lehet megformázni, a legtöbb operációs rendszer azonban csak bizonyos formátumokat tud elolvasni. A legtöbb másolás elleni rendszernek valamilyen szokatlan formátum az alapja, amely lehetetlenné teszi, hogy az operációs rendszer képes legyen az adatok olvasására és lemásolására. Különböző másolás elleni védelmi rendszerek közül választhatunk:

- Megváltoztathatjuk a szektorok sorrendjét. Ez úgy változtatja meg az elérési időt, hogy azt a védelmi rendszer is érzékelheti.
- Több szektort zsúfolhatunk egy sávra (512 bájtos szektoroknál a felső határ körülbelül 10 szektor).
- Egyszerűen kihasználhatunk egy szektorszámot.
- Egy szektornak szokatlan számot adhatunk (pl. $R=22$).
- Létrehozhatunk egy vagy több szokatlan méretű szektort.
- Hibás *C* és *H* értéket írhatunk fel.

A fenti technikák mindegyike felhasználható akár másolás elleni védelemre, akár a hajlékonylemez működési jellemzőinek megváltoztatására. A választástól függően elérhetjük, hogy egy szokásos módon megformázott lemez szokatlan tulajdonságai a DOS előtt titokban maradjanak, és a védelmi rendszert a szokásos módszerekkel észre se vegyük.

Az AT merevlemezes szolgáltatásai

AT 

Az AT-nál használt lemezmeghajtók annyira különböznek a többi modellnél használt meghajtóktól, hogy szükségessé teszik a többi új BIOS-lemezszolgáltatás bevezetését. Úgy tervezték őket, hogy képesek legyenek a nagykapacitású hajlékonylemezes meghajtók és az AT-nél használt sokféle merevlemez kiszolgálására. Ebben a részben csak vázlatosan ismertetjük az új szolgáltatásokat. Máskor is átsiklottunk a gépfüggő tulajdonságok miatt, most nem részletezzük e szolgáltatásokat, hiszen e könyvben fő célunk a teljes PC-családra érvényes általános elvek és a programozási gyakorlat ismertetése.

A 8-as szolgáltatás: a meghajtó aktuális paramétereinek olvasása

A 8-as szolgáltatás eredménye megadja a lemezmeghajtó jellemzőit. A *DL* a meghajtók számát (0, 1 vagy 2), a *DH* pedig a fejek vagy oldalak maximális számát tartalmazza. A *CH*-ban a cilinderek vagy sávok maximális száma, a *CL*-ben pedig a legnagyobb szektorszám található.

A 9-es szolgáltatás: a merevlemez paramétertáblázatainak inicializálása

A 9-es szolgáltatással a lemezek alaptáblázatát két merevlemezes konfigurációra lehet beállítani. A 65-ös és a 70-es megszakítás vektorának segítségével kapható meg a táblázatok címe. Ezt a szolgáltatást csak ún. „idegen” lemezmeghajtó üzembehelyezésekor kell használni.

A 10-es és 11-es szolgáltatás (hexa A és B): hosszú szektorok felírása és leolvasása

A 10-es szolgáltatás hosszú szektorokat olvas le, a 11-es pedig hosszú szektorokat ír fel a 20 Mbájtos merevlemezen. A hosszú szektor egy ún. *ECC*-t is tartalmaz. Ez a 4 bájtos hibakód lehetővé teszi, hogy a szektor adatainak hibátlanságáról nagy megbízhatósággal meggyőződjünk, ill. a hibákat javíthassuk.

A 12-es szolgáltatás (hexa C): a cilinder megkeresése

A 12-es szolgáltatás segítségével a lemez író-olvasó fejét egy megadott cilinderre pozicionálhatjuk. A *DL*-regiszterben van a meghajtó azonosítója, a *DH*-ban a fej száma, a *CH*-ban pedig a cilinder száma.

A 13-as szolgáltatás (hexa D): a lemez alapállapotba hozása (alternatív lehetőség)

A 13-as szolgáltatás újabb lehetőséget biztosít a merevlemez alapállapotba hozására. A meghajtó számát *DL*-ben kell megadni. A szolgáltatás a 0-s szolgáltatáshoz hasonlóan működik.

A 16-os szolgáltatás (hexa 10): a meghajtó készenléti állapotának ellenőrzése

A 16-os szolgáltatás ellenőrzi, készen van-e a merevlemezes meghajtó. A meghajtó számát a *DL*-regiszterben kell megadni, a státuszt az *AH*-ban kapjuk meg.

A 17-es szolgáltatás (hexa 11): a meghajtó újrakalibrálása

A 17-es szolgáltatás újrakalibrálja az adott rögzített lemezes meghajtót. A meghajtó számát a *DL*-regiszterben kell megadni, a státuszt az *AH*-regiszterben kapjuk meg.

A 20-as szolgáltatás (hexa 14): a meghajtó diagnosztikája

A 20-as szolgáltatás az AT lemezvezérlőjének belső diagnosztikai rutinját indítja el. A vezérlő állapotát az *AH*-regiszterben kapjuk.

A 21-es szolgáltatás (hexa 15): a lemez típusának leolvasása

A 21-es szolgáltatás az installált lemez típusát kérdezi le. Ha a meghajtó azonosítóját *DL*-ben adjuk meg, akkor az *AH*-ban a négyféle lemeztípus-azonosító egyikét kapjuk: ha *AH* értéke 0, ez azt jelenti, hogy nincs ilyen sorszámú lemezegység; ha *AH* értéke 1, ez olyan hajlékonyle-

mezes meghajtót jelent, ami nem érzékeli a lemez kicserélését (ilyen a legtöbb meghajtó); ha *AH* értéke 2, akkor a meghajtó érzékeli a lemez cseréjét (ilyenek az AT nagykapacitású meghajtói); végül a 3-as érték merevlemezre utal. Ha *AH* tartalma 3, akkor a *CX* és a *DX* 4 bájtos egészként tartalmazza a meghajtón található szektorok számát.

A 22-es szolgáltatás (hexa 16): a lemez státuszának megváltoztatása

A 22-es szolgáltatással a lemezváltást lehet lekérdezni, ha a meghajtók érzékelik a lemez cseréjét (ilyenek pl. az AT nagykapacitású meghajtói). Ha az *AH*-regiszter 0, akkor nem volt lemezcsere, ha 6, akkor volt. A *DL*-regiszterben kapjuk meg annak a meghajtónak a számát, ahol lemezcsere volt.

A 21-es és a 22-es szolgáltatás által biztosított lehetőség különösen olyan programok számára hasznos, amelyeknek érzékelni kell a lemezcserét. Vannak olyan kritikus lemezes műveletek (pl. a FAT leolvasása), ahol hasznos tudni, hogy volt-e lemezcsere. Ha igen, akkor esetleg törölni kell a lemezről a memóriában tárolt adatokat, és újra be kell azokat olvasni. Ha egy meghajtó nem érzékeli a cserét, akkor a programnak általában azt kell feltételeznie, hogy volt lemezcsere, és ennek megfelelően kell eljárnia; ez viszont a program hatékonyságának rovására megy. Ha lemezmeghajtókat vezérlő programot készítünk, akkor hasznos és hatékony, ha ellenőrizhetjük ezt az információt.

A 23-as szolgáltatás (hexa 17): a lemez típusának beállítása

A 23-as szolgáltatással az AT-nél beállíthatjuk a meghajtó és a lemez kombinációját. Ha *AL* értéke 0, akkor nincs meghajtó; ha *AL* értéke 1, akkor hagyományos lemez van a meghajtóban; ha *AL* értéke 3, akkor nagykapacitású lemez van a nagykapacitású meghajtóban. Ezt a szolgáltatást az 5-ös (formázó) szolgáltatással együtt a formázandó lemez típusának beállítására használjuk.

A lemez alaptáblázata

Egy lemezmeghajtó működését alapvetően az alaptáblázat néven ismert paraméterhalmaz szabályozza. A táblázat kiinduló változata ROM-ban van, az immár szabványosnak tekinthető F000 : EFC7 címen, de lehet új táblázatot is létrehozni. Az új táblázatot előbb a közönséges memóriában tároljuk, majd úgy változtatjuk meg az alaptáblázat megszakítási vektorát, hogy rámutasson. A 30-as megszakítás (hexa 1E) van fenntartva arra a célra, hogy az új táblázatra mutasson. A legősibb 1.00 változattól kezdve a DOS minden változata saját lemezének alaptáblázatát hozta létre, nem a ROM-ban tároltat használta. A lemez alaptáblázat a 10.5. táblázatban bemutatott 11 elemből áll. Bájtonként sorra vesszük az elemeket, és összevetjük a DOS 2.10 által használt értékeket az eredeti PC-nél használt kiinduló értékekkel. Az alaptáblázatban tárolt értékek nagy részének kevés hasznát vesszük, kivéve, ha a DOS által használt értékeket felül akarjuk írni.

A 0-s és 1-es bájt specifikus bájt. Ezek részét képezik a hajlékonylemezes vezérlőhöz kiküldött parancsláncnak. Az 1. bájt első 4 bitje a lépésidő. A ROM-BIOS ennyi időt hagy a

10.5 táblázat

A lemez alaptáblázat 11 bájtjának jelentése

Relatív cím	Jelentése
0	Az 1. specifikációs bájt: lépésidő
1	A 2. specifikációs bájt: DMA-mód
2	A művelet után a motor még ennyi ideig forog
3	Bájt/szektor 0=128; 1=256; 2=512; 3=1024
4	Az utolsó szektor száma
5	A szektorok közötti hézag mérete I/O utasításokkal
6	Az adat hossza, ha a szektor hossza nincs megadva
7	A szektorok közötti hézag mérete formázóműveleteknél
8	A formázott szektorokba írandó adat
9	Fejmegnyugvási idő
A	A motor felpörgetéséhez szükséges idő

meghajtónak, hogy a fej egyik sávról a másikra érjen. A kiindulási érték 8 milliszekundum minden sávra. A DOS 2.10 ezt 6 milliszekundumra csökkenti, ez gyorsítja a meghajtó működését. Az 1. bájtban található a DMA-mód beállítása, mert a PCjr-nek nem áll rendelkezésre a DMA a lemezes átviteleknél. Ezért a PCjr-lemez FAT-jában a DMA-mód bitje nincs beállítva. Furcsa módon a DOS 2.10 lemeztáblázatában ez a bit be van állítva, jelezvén, hogy van DMA. (Rejtély számunkra miért nem okoz ez gondot a PCjr-en.)

A 2-es bájt a 2-es relatív címen található. Tartalma azt szabja meg, hogy egy művelet után meddig járjon a meghajtó motorja. A motort bekapcsolva kell hagyni, hátha ismét szükség lesz rá. Az érték óraütések számaként van megadva: ez kb. 18 óraütés másodpercenként. A táblázat minden változatánál a bájt értéke 37 (hexa 25); ez azt jelenti, hogy a motor kb. 2 másodpercig jár.

A 3-as bájt a 3-as relatív címen kezdődik. Ez tartalmazza a szektor hosszának kódját. A kód megegyezik az 5-ös szolgáltatásnál megismerttel. (☛ L. a 10.4. ábrát). Értéke rendszerint 2, ez a szokásos 512 bájtos szektort jelenti. Az összes író, olvasó és verifikáló utasításnál a táblázatban a helyes hosszkódnak kell állnia, ez különösen akkor fontos, ha a hagyományostól eltérő méretű szektorral dolgozunk.

A 4-es bájt a 4-es relatív címen található. Ez a sávon szereplő utolsó szektor számát tartalmazza. Értéke a ROM-ba tárolt táblázatban 8, a DOS 2.10 táblázatban pedig 9.

Az 5-ös bájt az 5-ös relatív címen található. A szektorok közötti hézag méretét tartalmazza. Ezt adatok felírásakor és leolvasásakor használjuk. Valójában azt közli a ROM-BIOS-szal, hogy meddig várjon a következő szektor címjelének keresése előtt. Ezzel elkerülhetjük, hogy értelmetlen dolgot vizsgáljon a lemezen. A szabványos lemezek táblázatában ez az érték 42 (hexa 2A).

A 6-os bájt a 6-os relatív címen található. Az átvitt adatok mennyisége, értéke 255 (hexa FF). Ez a bájt szabja meg az átvitt adatmennyiség maximális hosszát, ha a szektor hossza nincs megadva.

A 7-es bájt a 7-es relatív címen van. Formázott sáv esetén ez tartalmazza a szektorok közötti hézag méretét. Ez természetesen nagyobb, mint az 5-ös címen található keresési hézag. Értéke rendszerint 80 (hexa 50).

A 8-as bájt a 8-as relatív címen található. Azt tartalmazza, hogy formázáskor milyen adat kerüljön az egyes szektorokba. A szokásos érték F6, az osztás jele. Ha kell, tetszőleges módon megváltoztatjuk.

A 9-es bájt a 9-es relatív címen található. A fej csillapodási idejét tartalmazza. A rendszer ennyit vár egy új sávra pozicionálás után, hogy a fej rezgése megszűnjön. A csillapodási idő kiindulási értéke 25 (hexa 19) milliszekundum, a DOS 2.10 ezt 15 (hexa F) milliszekundumra csökkenti.

A 10-es bájt az utolsó: a 10-es relatív címen található. A rendszer ennyi időt ad rá, hogy a motor felpörögjön. Egysége 1/8 másodperc. A beállított érték 4 vagyis 1/2 másodperc; a DOS 2.10 ezt 2-re csökkenti, ami 1/4 másodpercnek felel meg.

Jó mulatság a lemeztáblázat adataival játszani. Elég sok van belőlük ahhoz, hogy izgalmas játékot űzzünk, és sok galibát okozzunk. A következő program azt mutatja, hogyan lehet megváltoztatni egy szektorba beírható lemez típusát a lemez formázásakor. A szokásos F6 értéket AA-ra változtatjuk, pusztán azért, hogy a módszert szemléltessük.

```
10 DEF SEG
20 RELATIV=PEEK(120+0)+256*PEEK(120+1)
   ' a táblázat vektorcímének relatív része
30 SZEGMENS=PEEK(120+2)+256*PEEK(120+3)
   ' a táblázat vektorcímének szegmensrésze
40 DEF SEG=SZEGMENS
50 'ha a szegmens magas címen van, akor ROM-ban van, ezért nem változtatható meg
60 IF SZEGMENS > =&HF000 THEN PRINT
   "ROM-ban tárolt táblázatot használ."
70 FORMAT.ADAT=PEEK(RELATIV+8)            'régi adat
80 PRINT PRINT "A formátum adat" HEX$ (FORMAT.ADAT)
90 POKE RELATIV+8, &HAA                   'az adatot változtassuk hexa AA-ra
```

Megjegyzések és példák

A ROM-BIOS videoszolgáltatásokkal foglalkozó fejezetében nyugodtan tanácsolhattuk a ROM-BIOS-szolgáltatások közvetlen használatát, ha a helyzet úgy kívánta. A lemezes ROM-BIOS-szolgáltatásoknál azonban más a helyzet.

A programokban általában használt lemezes műveletekben a lemezek kezelését és felügyeletét legokosabb a DOS-ra bízni; vagy a programozási nyelvek hagyományos fájlkezelési műveleteit, vagy a DOS szolgáltatásait használva. (☛ L. a 14–18. fejezetet.) Ennek több oka is van. A legfontosabb ezek közül, hogy messze a legkönnyebb, ha a DOS-ra bízzuk a feladat elvégzését. A DOS szolgáltatásai szinte minden lemezes funkciót ellátnak, a lemez formátumának értelmezésétől az adatok átirányításáig (az ASSING-parancsig). Az esetek legnagyobb részében szükségtelen jobban elmerülni a rendszerszoftverben. Akadnak azonban olyan esetek, amikor rendkívül precízen kell dolgoznunk a lemez adataival. (Erre legtöbbször a másolás elleni védelem miatt van szükség.) Ilyenkor kell a ROM-BIOS szolgáltatásait használni.

Példáinkban Pascal nyelven írunk néhány szubrutint, amelyek abszolút lemezszektorokat írnak és olvasnak. Először definiáljuk az interfészt a Pascal nyelv szemszögéből. Ezt illusztrálja a következő program. Aki nem ismeri a Pascalt, és nincs kedve a szubrutint megfejteni, az nyugodtan hagyja ki. Az ezt követő assembly nyelvű interfész szubrutinnak akkor is jó hasznát veszi.

```
PROGRAM LEMEZ_INTERFACE;
{Ez definiálja a szegmens író – olvasó területét:}
TYPE
    SZEGMENS_TIPUS = ARRAY [0..511] OF BYTE;
VAR
    SZEGMENS_ADAT:SZEGMENS_TIPUS;
{Itt definiáljuk az assembly író – olvasó szubrutinokat:}
FUNCTION SZEGOLV (
                VAR S:SZEGMENS_TIPUS;           {adatterület}
                    D:INTEGER;                  {meghajtószám}
                    C:INTEGER;                  {sávszám}
                    H:INTEGER;                  {fejszám}
                    R:INTEGER )                 {szegmensszám}
        : BYTE;                                 {a státuszt bájtként kapjuk}
      EXTERNAL;
FUNCTION SZEGIR (
                  VAR S:SZEGMENS_TIPUS;         {adatterület}
                      D:INTEGER;                {meghajtószám}
                      C:INTEGER;                {sávszám}
                      H:INTEGER;                {fejszám}
                      R:INTEGER )               {szegmensszám}
        : BYTE;                                 {a státuszt bájtként kapjuk}
      EXTERNAL;
{Ez a rövid program beolvassa a lemezbetöltő rekordszegmenst.}
BEGIN
  IF SZEGOLV (SZEGMENS_ADAT, 0, 0, 0, 1) = 0
    THEN                                      {hibátlanul}
    ELSE ;                                    {hiba}
END.
```

E két szubrutin megismerése után már továbbléphetünk az assembly nyelvű interfész szubrutinokhoz. (☛ Az interfész szubrutinok formájával mindenki tisztában lehet, aki elolvasta az erre vonatkozó megjegyzéseket a 8. fejezetben, vagy tanulmányozta a 9. fejezetben található példát.)

Mindegyik példában egy egész sor új dolgot próbálunk bemutatni. Most két olyan különálló assembly nyelvű szubrutinnal van dolgunk, amit megpróbálunk egyesíteni. Ebben az esetben a két szubrutin azonos, bár más a nevük, és más ROM-BIOS-szolgáltatást használnak. Azt is figyeljük meg, hogy a veremből a paramétert [*BP*] + x alakú címhivatkozással vesszük ki. Ezen interfész esetében a veremben található első paraméter egy relatív cím, a többi tényleges érték (meghajtószám stb.). Annak ellenére, hogy az első paraméter egy cím, a többivel azonos módon kezeljük, mivel magára a címre vagyunk kíváncsiak, nem pedig a címen tárolt értékre. Amint a cím a birtokunkba kerül, egyszerűen átadjuk a ROM-BIOS-szolgáltatásnak.

Íme a szegmenst író és olvasó Pascal nyelvű interfész. Két, szinte teljesen azonos eljárásból áll: az egyik ír, a másik olvas.

```
    INTERFACE SEGMENT 'CODE'
              PUBLIC SZEGOLV
              PUBLIC SZEGIR
;Ez az olvasószolgáltatás
SZEGOLV       PROC        FAR
              PUSH        BP
              MOV         BP,SP
```

```
                PUSH    DS              ;DS áttöltése
                POP     ES              ;ES-be
                MOV     BX, [BP+14]     ;a relatív cím kivétele
                MOV     DL, [BP+12]     ;a meghajtó számának kivétele
                MOV     CH, [BP+10]     ;a sáv számának kivétele
                MOV     DH, [BP+08]     ;az oldal számának kivétele
                MOV     CL, [BP+06]     ;a szektor számának kivétele
                MOV     AL,1            ;1 szektort kérünk
                MOV     AH,2            ;olvasószolgáltatást kérünk
                INT     19              ;a lemezes szolgáltatás indítása
                MOV     AL,AH           ;a státusz beírása
                POP     BP
                RET     10              ;10 a paraméterek mérete a veremben
SZEGOLV         ENDP
; Ez az írószolgátatás (csak a szolgáltatás száma más).
SZEGIR          PROC    FAR
                PUSH    BP
                MOV     BP,SP
                PUSH    DS              ;DS áttöltése
                POP     ES              ;ES-be
                MOV     BX, [BP+14]     ;a relatív cím kivétele
                MOV     DL, [BP+12]     ;a meghajtó számának kivétele
                MOV     CH, [BP+10]     ;a sáv számának kivétele
                MOV     DH, [BP+08]     ;az oldal számának kivétele
                MOV     CL, [BP+06]     ;a szektor számának kivétele
                MOV     AL,1            ;1 szektort írunk
                MOV     AH,3            ;írói szolgáltatást kérünk
                INT     19              ;a lemezes szolgáltatás indítása
                MOV     Al,AH           ;a státusz beírása
                POP     BP
                RET     10              ;10 a paraméterek mérete a veremben
SZEGIR          ENDP
INTERFACE       ENDS
                END
```

11 A ROM-BIOS billentyűzet-szolgáltatásai

Bevezetés • 176

A billentyűzetszolgáltatások elérése • 176

A 0-s szolgáltatás: a következő karakter olvasása • 176
Az 1-es szolgáltatás: kész karakter jelzése • 177
A 2-es szolgáltatás: a váltóbillentyű állapotának jelzése • 177

Megjegyzések és példa • 178

Bevezetés

Bár a billentyűzetszolgáltatások sem bonyolultságokat, sem számukat tekintve nem vetekedhetnek a 9. fejezetben leírt képernyős vagy a 10. fejezetben ismertetett lemezes szolgáltatásokkal, fontosságuk miatt külön fejezetben foglalkozunk velük. A többi ROM-BIOS-szolgáltatást összegyűjtve ismertetjük a 12. fejezetben.

A billentyűzetszolgáltatások elérése

A billentyűzetszolgáltatásokat a 22-es (hexa 16) megszakítással hívjuk be. Három szolgáltatás tartozik ide; a 0-s, az 1-es és a 2-es. Akár a többi ROM-BIOS-szolgáltatásnál, itt is az *AH*-regiszterrel kell a megfelelőt kiválasztani.

11.1. táblázat
A ROM-BIOS billentyűzettel kapcsolatos három szolgáltatása

Szolgáltatás	Leírása
0	Olvasd be a következő karaktert!
1	Jelentsd, van-e kész karakter!
2	Közöld a váltóbillentyű állapotát!

A 0-s szolgáltatás: a következő karakter olvasása

A 0-s szolgáltatás megadja a billentyűzeten leütött következő karaktert. Ha a ROM-BIOS pufferében van kész karakter, akkor ezt azonnal megkapjuk. Ellenkező esetben a szolgáltatás megvárja az első kész karaktert. Mint a 6. fejezetben már szóltunk róla, minden karaktert egy bájtpár formájában kapunk meg. Az egyes bájtokat fő bájtnak, ill. kiegészítő bájtnak nevezzük. Az *AL*-regiszterben tárolt fő bájt értéke különleges karaktereknél 0 (ilyenek pl. a funkcióbillentyűk), közönséges billentyűknél pedig az ASCII-kód található itt. Az *AH*-regiszterben található kiegészítő bájt a különleges karakterek azonosítóját tartalmazza, szabvány PC-billentyűknél pedig a letapogatási kódot.

Ha a 0-s szolgáltatás meghívásakor a billentyűzet puffere üres, akkor a szolgáltatás megvárja az első karaktert – ezzel gyakorlatilag megbénítja a programot. A következő ismertetendő szolgáltatás lehetővé teszi, hogy a program leállítása nélkül megvizsgáljuk: van-e kész karakter.

Az egyes IBM Technical Reference kézikönyvek állításával ellentétben a 0-s és az 1-es szolgáltatás a közönséges ASCII-karakterekre és a különleges karakterekre is működik (pl. a funkcióbillentyűkre).

Az 1-es szolgáltatás: kész karakter jelzése

Az 1-es szolgáltatásból megtudhatjuk, van-e kész karakter. Nevezhetjük felderítő műveletnek is: ugyanis a karakter jelenlétéről értesülünk, amely mindaddig a ROM-BIOS billentyűzetpufferében marad, míg a 0-s szolgáltatással ki nem olvassuk. A zérusjelző (*ZF*) segítségével válaszol a szolgáltatás: az 1 azt jelzi, hogy nincs karakter, a 0 azt, hogy van. Ügyeljünk rá, nehogy megzavarjon minket a jelző látszólag fordított állása: a 0 azt jelenti, hogy van; az 1 azt, hogy nincs! Ha van karakter (*ZF*=0), akkor az *AL*-ben és az *AH*-ban meg is kapjuk. (A 0-s szolgáltatásnál már leírtuk.)

Ez a szolgáltatás két általánosan használt programozási technikánál különösen hasznos. Az egyiket *Vizsgálj és folytasd* módszernek nevezhetjük: ennél a program megvizsgálja, történt-e valami a billentyűzettel, és ha nem, akkor tovább kell futnia. Ezt általában akkor használják, ha egy folyamatosan futó programot egyetlen billentyű leütésével kell megszakítani. A másik ilyen művelet a billentyűzet pufferének kiürítése. Általában nagyvonalú megoldás, ha egy program lehetővé teszi, hogy „előre" gépeljünk: vagyis előre beírjuk a parancsokat. Vannak azonban olyan műveletek és helyzetek, ahol ez nem tanácsos. Ilyen a biztonsági okokból beiktatott ellenőrzési pont, pl.: „Minden rendben? Befejezhetem?". Ilyenkor szükség lehet rá, hogy egy program megtisztítsa a billentyűzet pufferét az ott tárolt adatoktól. A puffert pl. a 0-s és az 1-es szolgáltatással így üríthetjük ki:

```
10    'hívjuk meg az 1-es szolgáltatást, van-e kész karakter
20 WHILE ZF=0
30    'hívjuk meg a 0-ás szolgáltatást, hogy a karaktert eltávolítsa
40    'hívjuk meg az 1-es szolgáltatást, vizsgálja meg, van-e még karakter
50 WEND
```

Ellentétben néhány hivatkozási kézikönyv állításával, a 0-s és az 1-es szolgáltatás a közönséges ASCII-karakterekre és a különleges karakterekre egyaránt vonatkozik (pl. a funkcióbillentyűkre).

A 2-es szolgáltatás: a váltóbillentyű állapotának jelzése

A 2-es szolgáltatás az *AL*-regiszterben megadja a váltóbillentyű (*Shift*) állapotát. Ezt a státuszt bitről bitre átveszi a hexa 417-es címen tárolt 1. billentyűzet státuszbájtból. A 11.2. táblázatban

11.2. táblázat
Az AL-regiszterben visszakapott státuszbitek a 2-es szolgáltatásnál

Bit 7 6 5 4 3 2 1 0	Jelentése
X	Inzertbillentyű: 1 = aktív
. X	Nagybetűzár (Caps Lock): 1 = aktív
. . X	Számzár (Num Lock): 1 = aktív
. . . X	Képernyőzár (Scroll Lock): 1 = aktív
. . . . X . . .	Alt-váltó: 1 = aktív (Alt lenyomva)
. X . .	Ctrl-váltó: 1 = aktív (Ctrl lenyomva)
. X .	Közönséges váltó: 1 = aktív (bal oldali Shift lenyomva)
. X	Közönséges váltó: 1 = aktív (jobb oldali Shift lenyomva)

látható az egyes bitek jelentése. (☛ A 6. fejezetben már ismertettük a hexa 418 és a hexa 488-as címen tárolt többi státuszbájtot.)

Általában a 2-es szolgáltatás és a státuszbit információi nem különösebben hasznosak. Ha valaki mutatósabb billentyűzetkezelő programot akar írni, mégis szüksége lehet rá. Gyakran alkalmazzák olyan programokban is, amelyek valamilyen szokatlan dolgot művelnek a billentyűkkel: pl. megkülönböztetik a jobb és a bal oldali *Shift*-billentyűt.

Megjegyzések és példa

Ha valaki választhat a programozási nyelv és a ROM-BIOS billentyűzetszolgáltatásai között, akkor nyugodtan választhatja akármelyiket. Bizonyos esetekben ugyan nyomós érvek szólnak a ROM-BIOS-szolgáltatások közvetlen használata ellen (pl. a lemezes szolgáltatásoknál), ezek az érvek a billentyűzettel kapcsolatos szolgáltatásoknál nem ilyen erősek. Azonban itt is felhívjuk a figyelmet rá, hogy a DOS-szolgáltatások által nyújtott lehetőségeket alaposan meg kell vizsgálnunk, mielőtt a BIOS-szolgáltatásokhoz folyamodnánk. Lehet, hogy a DOS is minden szükségeset megad, és a DOS-szolgáltatások hosszabb életűek lesznek a személyi számítógépek örökké változó világában.

A legtöbb programozási nyelv billentyűzettel kapcsolatos műveletei a DOS szolgáltatásaira épülnek: ez a tény határozott előnyökkel jár. Az elsődleges előny, hogy a DOS-szolgáltatások lehetővé teszik a bemenet szerkesztését a DOS szokásos szerkesztési műveleteivel – ez mindaddig lehetséges, amíg a *Return*-billentyűt le nem nyomtuk. Ha feltesszük, hogy programunknak nincs szüksége arra, hogy maga felügyelje az adatok bevitelét, igen sok programozási munkát (és a felhasználó felkészítését) takaríthatunk meg, ha közvetlenül vagy közvetve (a programozási nyelven keresztül) a DOS szolgáltatásait vesszük igénybe. Ha teljesen ellenőrzésünk alatt akarjuk tartani a billentyűzetet, akkor végül mégis a ROM-BIOS-szubrutinok használatánál kötünk ki. A döntés a mi kezünkben van.

A billentyűzetszolgáltatások használatát bemutató assembly nyelvű példánkban még nagyobb fába vágjuk fejszénket: egy puffer teljes kiürítését mutatjuk be. A szubrutin azt a feladatot látja el, amit az 1-es szolgáltatásnál vázoltunk.

A pufferürítő szubrutinban bemutatott újdonságok között szerepel a címkék és az ugróutasítások használata. Az assembly nyelvű szubrutinok általános ismertetésénél (a 8. fejezetben) már említettük, hogy bizonyos körülmények között szükség van az **ASSUME CS**-utasításra: itt ezt működés közben láthatjuk.

Az **ASSUME CS**-utasítás használatakor tulajdonképpen mindig félrevezetjük az assemblert a *CS*-regiszter tartalmát illetően, mivel a *CS* nem feltétlenül oda mutat, ahova mondjuk. Ez a kis félrevezetés teljesen veszélytelen mindaddig, amíg az összes ugróutasítás (mint a példában szereplő **JZ**) rövid ugrás. Ennek az az oka, hogy a rövid ugrás az utasításszámláló aktuális értékéhez viszonyított relatív vezérlésátadás; ezt a *CS:IP* kombinációval jelöljük. Végrehajtásakor az assembler az *IP*-hez – nem a *CS*-hez – viszonyított vezérlésátadást generál, ezért igazában nem is kell a *CS*-ről sokat tudnia. Valójában arról van szó, hogy azt mondjuk az assemblernek: feltételezzen valamit (**ASSUME**) a *CS*-regiszterről, csupán azért, mert az assembler túl nehézkes. Íme egy példa:

```
;KBCLEAR üríti a billentyűzet pufferét
INTERFACE   SEGMENT     'CODE'
            PUBLIC      KBCLEAR
            ASSUME      CS:INTERFACE
```

```
KBCLEAR     PROC    FAR
            PUSH    BP
            MOV     BP,SP
            MOV     AH,1              ; az első adatellenőrzés
            INT     22
WHILE:
            JZ      SHORT RETURN      VISSZA
                                      ; amíg ZF=0
            MOV     AH,0
                                      ; dobd el az adatot
            INT     22
            MOV     AH,1              ; ismételd az adat ellenőrzését
            JMP     SHORT WHILE
RETURN:                               ; vissza a hívóhoz
            POP     BP
            RET
KBCLEAR     ENDP
INTERFACE   ENDS
            END
```

12 Egyéb szolgáltatások

Bevezetés • 182

Az RS – 232 soros kommunikációs szolgáltatások • 182

A 0-s szolgáltatás: a soros port paramétereinek inicializálása • 183
Az 1-es szolgáltatás: egy karakter kiküldése • 184
A 2-es szolgáltatás: egy karakter vétele • 184
A 3-as szolgáltatás: a soros port állapotának lekérdezése • 185

A magnós szolgáltatások • 186

A 0-s szolgáltatás: a motor bekapcsolása • 186
Az 1-es szolgáltatás: a motor kikapcsolása • 187
A 2-es szolgáltatás: adattömbök olvasása • 187
A 3-as szolgáltatás: adattömbök írása • 187

Kiterjesztett szolgáltatások az AT-hez • 188

A nyomtatás szolgáltatásai • 188

A 0-s szolgáltatás: egy karakter küldése a nyomtatóhoz • 189
Az 1-es szolgáltatás: a nyomtató inicializálása • 189
A 2-es szolgáltatás: a nyomtató állapotának lekérdezése • 189

Egyéb szolgáltatások • 190

Az 5-ös megszakítás (hexa 5): a képernyőtartalom kinyomtatása • 190
A 17-es megszakítás (hexa 11): a perifériák listázása • 191
A 18-as megszakítás (hexa 12): a memória méretének jelzése • 192
A 24-es megszakítás (hexa 18): a BASIC betöltése • 192
A 25-ös megszakítás (hexa 19): az indító (bootstrap) betöltése • 192
A 26-os megszakítás (hexa 1A): időszolgáltatások • 193

Bevezetés

Ebben a fejezetben azokat a ROM-BIOS szolgáltatásokat tekintjük át, amelyeket önálló fejezetként nem tárgyalunk. Ezek az RS – 232 soros kommunikációs szolgáltatások, a magnós szolgáltatások, az AT kibővített szolgáltatásai, a nyomtatás szolgáltatásai és még néhány különleges szolgáltatás.

Az RS – 232 soros kommunikációs szolgáltatások

Ez a fejezet tárgyalja a ROM-BIOS-ban lévő RS – 232 aszinkron soros kommunikációs port szolgáltatásait. A ROM-BIOS szolgáltatások részletes leírása előtt meg kell ismernünk a soros kommunikációs port terminológiáját. Feltételezzük az adatátvitel alapjainak ismeretét, akinek az egyes kifejezések ismeretlenek volnának, kérjük nézzen utána egy adatátvitellel foglalkozó szakkönyvben.

A számítógép RS – 232 vonalának számos elnevezése van. Az egyik legelterjedtebb a port (kapu), bár az előző fejezetekben nem így használtuk. E könyv legnagyobb részében a port szó a 8088-as mikroprocesszor által használt címezhető utat jelentette, amellyel a számítógép áramköri korlátain belül a számítógép egyéb részeivel tartott fenn kapcsolatot. Portszámokon az **INP** és az **OUT** BASIC utasításokra és az assembler-szintű **IN** és **OUT** műveletek révén elérhető címezhető portok címét értjük. Ettől eltér az RS – 232 aszinkron soros kommunikációs port, mert ez egy általános célú I/O útra vonatkozik, amelyet többféle, a számítógépen kívüli információfeldolgozó berendezés összekapcsolására lehet használni. A soros portokat a PC-kben elsősorban telekommunikációra használják (ezalatt a modemen keresztüli telefon-összeköttetést is értve), de alkalmazzák a soros típusú nyomtatókhoz való adatátvitelre is.

A soros kommunikációs szolgáltatásokat a 20. (hexa 14) megszakításon keresztül lehet elérni. Négy szolgáltatás közös az összes IBM-típusban. Ezeket 0-tól 3-ig számozzuk, és az *AH*-regiszteren keresztül választjuk ki (l. 12.1. táblázat).

Az IBM személyi számítógépek eredeti terve hét soros port bővítését tette lehetővé, bár egy számítógép általában egynél vagy kettőnél többet nem használ. Függetlenül attól, hogy hány soros port van, a port sorszámát a *DX*-regiszterben adjuk meg. Ha csak egyetlen soros porttal rendelkezünk, ennek adapterét a *DX*-regiszterben 0 jelöli.

12.1. táblázat
A 20-as (hexa 14) megszakításon keresztül elérhető négy RS – 232 soros portra vonatkozó szolgáltatás

Szolgáltatás	Leírása
0	A soros port paramétereinek inicializálása
1	Egy karakter kiküldése
2	Egy karakter vétele
3	A soros port állapotának lekérdezése

A 0-s szolgáltatás: a soros port paramétereinek inicializálása

A 0-s szolgáltatás beállítja a különböző RS – 232 paramétereket, és inicializálja a soros portot. Négy paramétert állít be: az adatátviteli sebességet, a paritást, a stopbitek számát és a karakterméretet (szóhossznak is nevezik). A paramétereket egy 8 bites kódba fogja össze, amelyek a

12.2. táblázat
A soros port négy paraméterének bitbeállítása

Adatátviteli sebesség		
Bit 7 6 5	Érték	Bit/másodperc (baud)
0 0 0	0	110
0 0 1	1	150
0 1 0	2	300
0 1 1	3	600
1 0 0	4	1200
1 0 1	5	2400
1 1 0	6	4800
1 1 1	7	9600

Paritás		
Bit 4 3	Érték	Jelentés
0 0	0	Semmilyen
0 1	1	Páratlan
1 0	2	Semmilyen
1 1	3	Páros

Stopbitek száma		
Bit 2	Érték	Jelentés
0	0	Egy
1	1	Kettő

Karakterméret		
Bit 1 0	Érték	Jelentés
0 0	0	Nem használt
0 1	1	Nem használt
1 0	2	7 bites*
1 1	3	8 bites

* Csak 128 szabvány ASCII-karakter van, úgyhogy ezeket lehet 7 bites karakterekként és az elterjedtebb 8 bites karakterként is továbbítani.

12.3. táblázat

A soros port paramétereinek bitsorrendje a 0. szolgáltatás visszajelzésekor az Al-regiszterben

Bit 7	6	5	4	3	2	1	0	Felhasználás
X	X	X	.	.	.	.	.	Az átviteli sebesség kódja
.	.	.	X	X	.	.	.	Paritáskód
.	.	.	.	.	X	.	.	Stopbitkód
.	.	.	.	.	.	X	X	A karakterméret kódja

12.3. táblázaton látható kiosztásban kerülnek az *AL*-regiszterbe. Az egyes kódok bitbeállítását a 12.2. táblázat mutatja. A szolgáltatás befejeztével az *AX* a 3-as szolgáltatásnál már megismert módon jelzi a kommunikációs állapotot. (L. részletesebben a 3-as szolgáltatásnál.)

📖 MEGJEGYZÉS: A modemeket használó személyi számítógépek legtöbbjénél a 300 baudos sebességet használták általában, ez viszont meglehetősen lassú. Mostanában az 1200 baudos sebesség a legelterjedtebb, különösen olyan soros alkalmazásoknál, amelyek gyors átvitelt igényelnek. Valószínű azonban, hogy ez a sebesség eléri majd a 2400 baudot is. [JR] A PCjr maximális átviteli sebessége 4800 baud. Ha 9600 baudra próbáljuk beállítani, akkor is csak 4800 bauddal ad. (Modem nélkül 9600 baudos átviteli sebességet is elérhetünk. A nagyobb sebesség ára a kisebb, néhányszor 10 méteres áthidalható távolság. A lektor.)

Az 1-es szolgáltatás: egy karakter kiküldése

Az 1-es szolgáltatás egy karaktert küld ki a soros portra. A karakter az *AL*-ből veszi, és az *AH*-ban jelzi vissza a szolgáltatás eredményességét. Ha $AH=0$, akkor a szolgáltatás eredményes volt. Ha nem, az *AH* 7. bitje jelenti a hibát, és az *AH* többi bitje a hiba típusát. Ezeket a biteket a 3-as szolgáltatásnál tárgyaljuk.

E szolgáltatás révén kapott hibajelentésnek van egy gyengéje. Mivel a 7. bit a hiba előfordulását jelzi, ez a bit nem mutathatja az időtúlfutási (time-out) hibát (ahogy ezt a 3-as szolgáltatás esetén elvárjuk). Következésképpen, amikor ez a szolgáltatás vagy a 2-es szolgáltatás jelez hibát, a legegyszerűbb és a legmegbízhatóbb a hiba természetének ellenőrzésére a 3-as szolgáltatás által adott teljes státuszjelzés használata.

A 2-es szolgáltatás: egy karakter vétele

A 2-es szolgáltatás vesz egy karaktert a *DX*-ben meghatározott kommunikációs vonalról, és azt az *AL* regiszterbe helyezi. Ez a szolgáltatás olyan karakterre vagy jelre vár, amely a szolgáltatás befejezését jelzi, mint pl. az időtúlfutás. Az *AH* 7. bitje mutatja a szolgáltatás sikerét vagy sikertelenségét, ahogy ezt az 1-es szolgáltatásnál elmondtuk. Vegyük figyelembe az 1-es szolgáltatásnál a hibakezelésre vonatkozó tanácsunkat, és nézzük meg a 3-as szolgáltatás hibakódját.

A 3-as szolgáltatás: a soros port állapotának lekérdezése

A 3-as szolgáltatás a teljes soros port állapotkódját az *AX* regiszterbe helyezi. A 16 bit mindegyike egyenként egy-egy lehetséges hibát jelent. Az állapotbiteket két csoportba soroljuk: az *AH*-beliek a vonali állapotát jelzik (ezeket az 1-es és a 2-es szolgáltatásnál fellépő hiba esetén is jelzik), az *AL*-beliek pedig a modem állapotát jelzik (ha ez szükséges). A 12.4. táblázat az állapotbitek kódolását tartalmazza. Szembetűnő, hogy a kódok némelyike a hibát jelzi, míg más kódok egyszerűen csak állapotot jelentenek.

🕮 MEGJEGYZÉS: Meg kell említenünk az időtúlfutási hiba (*AH*, bit 7) egy különlegességét: az eredeti PC számára készített ROM-BIOS legelső változatában volt egy programhiba, amely következtében a soros port időtúlfutási hibáját átviteli (shift – regiszter – üres vagy megszakítás – észlelte) hibaként jelezte (10000000 bitek helyett 01010000). Az ezt követő ROM-BIOS változatokban ezt a hibát már kijavították, mégis sok kommunikációs program ezeket a hibakódokat szkeptikusan kezeli. Érdemes ezt megjegyezni. (☛ L. a ROM-BIOS változat dátumának és gépazonosító kódjainak értelmezését.)

12.4. táblázat
A 3. szolgáltatás által az AX-regiszterbe helyezett állapotbájtok bitkódolása

Bit 7 6 5 4 3 2 1 0	Jelentése (ha értéke 1)
AH-regiszter (vonalállapot)	
1	Időtúlfutás (time-out hiba)
. 1	Az átviteli shiftregiszter üres
. . 1	Az átviteli tárolóregiszter üres
. . . 1	A kapcsolat megszakításának érzékelése
. . . . 1 . . .	Kerethiba
. 1 . .	Paritáshiba
. 1 .	Túlcsordulási hiba
. 1	Az adat kész
Bit 7 6 5 4 3 2 1 0	**Jelentése (ha értéke 1)**
AL-regiszter (modem állapot)	
1	Vonali hiba érzékelése
. 1	Gyűrűkapcsolás vége
. . 1	Adatkészlet (modem) készenlétben
. . . 1	Adásrakész jel
. . . . 1 . . .	Delta fogadóvonal jele érzékelve
. 1 . .	A jeláram felfutóéle érzékelve
. 1 .	Delta adatkészlet készenlétben
. 1	Adásrakész delta jel

A magnós szolgáltatások

A magnós szolgáltatásokat az ilyen perifériával kiegészített számítógépeknél használjuk. A magnó része néhány PC típusnak, pl. az eredeti PC-nek, a PCjr-nek, de nem része az XT-nek, a hordozható PC-nek, az AT-nek, a 3270-PC-nek és a PC család néhány más tagjának. A kazettaport tehát többé-kevésbé elavult, de amikor az eredeti PC-t tervezték, feltételezték, hogy lehet igény rá. Nem volt, ezért a szolgáltatás majdnem teljesen kihasználatlan maradt. A kazettaporttal adatokat és programokat (elsősorban BASIC-programokat) kívántak rögzíteni normál audio kazettás magnószalagon ugyanúgy, mint számos más számítógépen. Azonban a kereskedelemben még sohasem találkoztunk szalagra rögzített PC programmal. A kazettaport talán egyetlen ismert felhasználása nem egyéb, mint egy házilag barkácsolt soros port. Az IBM mégis támogatja a kazettaport használatát, az itt tárgyalt ROM-BIOS szolgáltatásoknál és a BASIC-nél, mert lehetővé teszi számunkra, hogy szalagra írjunk, ill. onnan olvassunk adatokat vagy BASIC-programokat.

12.5. táblázat
A 21 (hexa 15) megszakítás révén elérhető négy magnós szolgáltatás

Szolgáltatás	Megnevezés
0	A magnó motorjának bekapcsolása
1	A magnó motorjának kikapcsolása
2	Adattömbök olvasása
3	Adattömbök írása

Itt jegyezzük meg, hogy a kazettaport bármely felhasználásával velejárnak bizonyos problémák. Először: nincs meg az összes PC modellen ez a port. Másodszor: kevés PC-t szereltek fel megfelelő kábelcsatlakozással, pedig a magnókazettával való munkához erre szükség lenne. Harmadszor: a magnókazetta használata lényegesen több manuális beavatkozást kíván, mint ahogy azt várnánk. A szalag visszatekercselését pl. nem tudja a program végrehajtani.

A kazettaszolgáltatásokat a 21-es megszakításon át lehet elérni. Négy szolgáltatás áll rendelkezésünkre, amelyek 0-tól 3-ig számozottak. A szolgáltatást minden esetben az *AH* regiszter értéke választja ki (l. a 12.5. táblázatot).

A 0-s szolgáltatás: a motor bekapcsolása

A 0. szolgáltatás kapcsolja be a magnó motorját, ez a lemezes szolgáltatásokkal ellentétben nem automatikus művelete a ROM-BIOS szolgáltatásoknak. A szolgáltatást használó bármely programnak fel kell készülnie a késleltetésre: meg kell várnia, hogy a motor elinduljon.

Az 1-es szolgáltatás: a motor kikapcsolása

Az 1-es szolgáltatás kikapcsolja a magnó motorját, ez sem automatikus művelete a ROM-BIOS szolgáltatásoknak.

A 2-es szolgáltatás: adattömbök olvasása

A 2-es szolgáltatás egy vagy több adattömböt olvas. A magnóról az adatokat szabvány méretű, 256 bájtos tömbökben mozgatja ugyanúgy, mint ahogy a lemezes adatok általában a szabványos 512 bájtos méretet használják. Az olvasandó bájtok számát a *CX* regiszterbe kell helyezni. Bár az adatokat a szalagon 256 bájtos tömbökbe helyezzük, bármilyen számú bájt írható vagy olvasható. Következésképpen a *CX* regiszterbe helyezett bájtszámnak nem kell feltétlenül 256 egész számú többszörösének lennie.

Az *ES*: *BX* regiszterpárt mutatónak használjuk azon memóriaterület kijelölésére, ahová az adatokat helyezni kell.

12.6. táblázat
A 2. szolgáltatás által az AH-regiszterben visszaadott hibakódok, ha a CF hibát jelez az adattömbök olvasásánál

Kód	Jelentés
1	Ciklikus redudancia-ellenőrzési hiba (CRC)
2	Elvesztett adatátvitel: a bitjelek összekeveredése
3	Nem talált adatot a szalagon

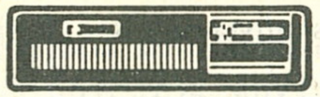
A szolgáltatás befejezése után a *DX* a ténylegesen beolvasott bájtok számát tartalmazza, az *ES* : *BX* arra a bájtra mutat, amely az utolsó átvitt bájt után helyezkedik el; a carry flag (*CF*) 0 vagy 1 értéke pedig a művelet sikerét vagy a sikertelenségét mutatja. Hiba esetén az *AH* jelzi a hiba természetét. A 12.6. táblázat mutatja be a hibakódokat.

A 3-as szolgáltatás: adattömbök írása

A 3-as szolgáltatás az egyenként 256 bájtos adattömbökből egyet vagy többet ír a szalagra (l. a 2-es szolgáltatást). Ugyanúgy, mint a 2-es szolgáltatásnál, a *CX* regiszterben adhatjuk meg a kért bájtok számát, és az *ES*: *BX* jelöli ki az adat helyét a memóriában. Ha a beírt adat mennyisége nem többszöröse 256 bájtnak, akkor az utolsó tömböt teljes mértékűre egészíti ki.

Miután a szolgáltatás befejeződött, a *CX*-et nullára kell csökkenteni, és az *ES*: *BX*-nek az utolsó memóriába írt bájt mögé kell mutatnia. Furcsa módon ezen szolgáltatásra nincsenek hibajelek megadva. Lényegében a magnó nem alkalmas arra, hogy a számítógépet tájékoztassa bármilyen nehézségről. Ez arra kényszeríti a ROM-BIOS-t, hogy az adatokat a művelet sikerességében vakon bízva írja ki. Szükségtelen hangsúlyozni, hogy ellenőrzésképpen nem volna rossz ötlet visszaolvasni bármely kiírt adatot.

Kiterjesztett szolgáltatások az AT-hez

AT 

Az AT-vel számos új BIOS szolgáltatást vezettek be a bővített memória és néhány további fejlesztés támogatására. Ezeket a 21-es megszakításon keresztül hívhatjuk ugyanúgy, mint a kazetta I/O szolgáltatásokat. A szolgáltatást kiválasztó szám (hexa 80–91) az *AH* regiszterben van. (☛ További információk érdekében l. a 13. fejezetet és az AT Technical Reference kézikönyvét.)

12.7. táblázat
A 21. (hexa 15) megszakítás révén elérhető 12 kiterjesztett AT-szolgáltatás

Szolgáltatás, hexa	Megnevezés
80	A készülék nyitva
81	A készülék zárva
82	A program befejezése
83	Eseményvárás
84	Botkormány (joystick) van
85	A *SysReq*-gomb lenyomva
86	Várakozás
87	Tömbmozgatás
88	Bővített memória méretének lekérdezése
89	Kapcsolás a virtuális memóriához (Figyelmeztetés: használat előtt nézze meg a BIOS listáját!)
90	A „készülék foglalt” hurok
91	Jelzőbit (flag) beállítása és a megszakítás végrehajtása

A nyomtató szolgáltatásai

A ROM-BIOS nyomtatószolgáltatások a nyomtató kimenetet kezelik. A szabvány PC világában ezek szigorúan a párhuzamos nyomtatóadapterre vonatkoznak. Néhány PC-típuson mindazonáltal a nyomtató kimenetét automatikusan át lehet irányítani a sorosportra. Ezt a szolgál-

12.8. táblázat
A 23. (hexa 17) megszakítás révén elérhető három nyomtatószolgáltatás

Szolgáltatás	Leírás
0	Egy karakter küldése a nyomtatóhoz
1	A nyomtató inicializálása
2	A nyomtató állapotának lekérdezése

tatást a PCjr ROM-BIOS-a nyújtja. A ROM-BIOS nyomtatószolgáltatásokat a 23-megszakítás kezeli. Három 0-tól 2-ig számozott szolgáltatás van. Az *AH* regiszteren keresztül kérhetjük őket (l. a 12.8. táblázatot). Az általános PC család tervezése lehetővé teszi egy vagy több nyomtató installálását, ezért az összes szolgáltatáshoz a nyomtató sorszámát meg kell határozni a *DX* regiszterben. A 0 nyomtatószámot automatikusan használja a print-screan szolgáltatás.

A 0-ás szolgáltatás: egy karakter küldése a nyomtatóhoz

A 0. szolgáltatás egy karaktert küld a nyomtatóhoz, a nyomtatandó bájtot az *AL*-be helyezve. Amikor a szolgáltatás befejeződik, az *AH* a nyomtató státuszát mutatja (l. a 2-es szolgáltatást); ezzel a művelet sikeres vagy sikertelen voltát határozhatjuk meg. (L. a 2-es szolgáltatást, amelyet szintén a művelet sikerének vagy meghiúsulásának meghatározására használhatunk; és a külön megjegyzéseket a 2-es szolgáltatásnál az időtúlfutási hibára vonatkozóan.)

Az 1-es szolgáltatás: a nyomtató inicializálása

Az 1-es szolgáltatás inicializálja a nyomtatót. Ennek érdekében a szolgáltatás egyszerűen két vezérlőkódot (hexa 08 és 0C) küld a nyomtatóportra (rendszerint a 762, a hexa 2FA port), és ugyanúgy, mint a másik két szolgáltatás, a nyomtató állapotát az *AH*-ban jelzi.

A 2-es szolgáltatás: a nyomtató állapotának lekérdezése

A 2-es szolgáltatás jelzi a nyomtató állapotát az *AH* regiszterben. Az egyes bitkódokat a 12.9. táblázat mutatja.

A nyomtató időtúlfutási hibája sok nehézséget okozott az IBM személyi számítógépekben. Bármelyik I/O meghajtónak be kell állítania az általa vezérelt berendezés válaszidejének korlátját. Ideális esetben ez az időkorlát azért nem lehet túl hosszú, hogy a válaszképtelen berendezést időben jelenteni lehessen. Sajnos, van egy olyan nyomtatóművelet, amely meglepő-

12.9. táblázat
A 2. szolgáltatás által az AH-regiszterben visszaadott állapotbitek jelentése

Bit 7 6 5 4 3 2 1 0	Jelentés (amikor értéke 1)
1	A nyomtató nem foglalt (0 = foglalt)
. 1	Nyugtázás a nyomtatóról
. . 1	Papírhiány jelzése
. . . 1	Ez a kiválasztott nyomtató
. . . . 1 . . .	I/O hiba
. 1 . .	Nem használt
. 1 .	Nem használt
. 1	Időtúlfutási (time-out) hiba

en hosszú időt vehet igénybe: a lapdobás (vagyis a jelenlegi pozíciótól a következő oldal tetejéig való lépés). Az erre megengedett idő a ROM-BIOS egyes változatainál eltérő, ezért az időtúlfutási hibajelet óvatosan kell kezelni.

Egyéb szolgáltatások

Most pedig eljutottunk az összes, máshova nem sorolható ROM-BIOS szolgáltatásokhoz: ezek közül néhánnyal az IBM szándéka az volt, hogy használják és néhánnyal az (elsősorban azzal a szolgáltatással, amely kocsivissza-soremelés karakter kombinációt küld a monitor képernyőjére), hogy ne használjuk (nem is ajánljuk ezeket). Ebben a részben a 12.10. táblázaton bemutatott hat megszakítást nézzük át egyenként.

12.10. táblázat
Az IBM által támogatott hat egyéb ROM-BIOS-szolgáltatás és a hozzájuk társuló megszakítások

Megszakítás		Leírás
Dec	Hex	
5	5	Képernyő-kinyomtatási szolgáltatás rutinja
17	11	Berendezéseket felsoroló szolgáltatás
18	12	Memóriaméret-szolgáltatás
24	18	A ROM-BASIC nyelv aktiválása
25	19	A beindító (bootstrap) rutin aktiválása
26	1A	Időszolgáltatások

Az 5-ös megszakítás (hexa 5): a képernyőtartalom kinyomtatása

Az 5-ös megszakítás aktiválja a képernyő tartalmát kinyomtató szolgáltatást. Ezt a *Shift-PrtSc* kombinációra való reagálásként a billentyűzetet kezelő rutin használja. Bármely program, amely a képernyő tartalmát óhajtja kinyomtatni, biztonságosan és kényelmesen teheti ezt az 5-ös megszakítás meghívásával. A képernyőtartalom szubrutint kivételesen megszakításvezérléssel készítették azért, hogy a szolgáltatást saját programjainkba is be tudjuk építeni.

Az 5-ös szolgáltatás kezeli a kurzor aktuális pozícióját a képernyőn, és kinyomtat bármely nyomtatható karaktert a képernyőről mind szöveges, mind grafikus üzemmódban. A szabványos képernyős szolgáltatásokat használja (azokat, amelyek végigfuttatják a kurzort a képernyőn, és kiolvassák a karaktereket a képernyő tárolójából), és használja a szabványos nyomtatószolgáltatásokat is.

Ez a szolgáltatás az összes kimenetét a 0-s számú nyomtatóhoz irányítja, amely az alapnyomtató. Ehhez a szolgáltatáshoz nem tartozik sem bemeneti, sem kimeneti regiszter. Egy állapotkód azonban megtalálható a hexa 500 alacsony memóriacímen (☛ l. 3. fejezet). Ha ezen a helyen levő bájt értéke 255 (hexa FF), akkor az előző kinyomtatás nem volt sikeres. A 0 érték

jelzi azt, hogy nem volt hiba: az 5-ös megszakítás kész az indításra; az 1 érték pedig azt mutatja, hogy a képernyőnyomtatás éppen folyamatban van. Bármely ilyen kérés másodszorra már hatástalan.

A 17-es megszakítás (hexa 11): a perifériák listázása

A 17-es megszakítás felsorolja a számítógéphez kapcsolt berendezéseket. Megegyezik a hexa 410 memóriacímen tárolt információval (☛ l. 3. fejezet). A visszajelzés 16 bites kódjai a 12.11. táblázatban láthatók. A visszajelzés az *AX* regiszterben található (l. a 18-as megszakítást a kiegészítő szolgáltatásokra vonatkozóan).

12.11. táblázat

Az AX-regiszterben jelzett és a 17. (hexa 11) megszakítással elérhető berendezéslista bitkódolása

Bit																Jelentése
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
X	X	.	.	.	.	.	.	.	.	.	.	.	.	.	.	Az installált nyomtatók száma
.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	.	Soros nyomtató: 1 = installálva, (csak a PCjr-nél; a PC-nél, és az XT-nél nem)
.	.	.	X	.	.	.	.	.	.	.	.	.	.	.	.	Játékadapter: 1 = installálva (mindig igaz a PCjr-re)
.	.	.	.	X	X	X	.	.	.	.	.	.	.	.	.	Az RS – 232 soros portok száma
.	.	.	.	.	.	.	X	.	.	.	.	.	.	.	.	A DMA chip: 0 = installálva (1 = nincs installálva – rendszerint a PCjr-ban)
.	.	.	.	.	.	.	.	X	X	.	.	.	.	.	.	+ 1 = a hajlékonylemezes meghajtók száma: 0 = 1 meghajtó (l. 0 bit)
.	.	.	.	.	.	.	.	.	.	X	X	.	.	.	.	Kezdeti videoüzemmód: 10 = 80-oszlopos színes, 11 = monokrom (PC vagy XT); 01 = 40-oszlopos PCjr
.	.	.	.	.	.	.	.	.	.	.	.	X	X	.	.	Rendszerkártyán levő RAM: 11 = 64 K (normál esetben az összes típusra vonatkozóan)
.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	.	nem használt (0-ra állítva)
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	X	1, ha van hajlékony mágneslemezes meghajtó (ha ez fennáll, l. a 6. és 7. biteket)

A berendezés az információt a lehető legnagyobb pontosság elve alapján állítja össze, mégsem biztos, hogy hibátlanul. A különböző típusok eltérő módszert alkalmaznak az információ szerzésére. A berendezések listáját csak egyszer – a bekapcsoláskor – állítja össze, azután az a memóriában marad. Ez azt jelenti, hogy a berendezéslistát programból változtatni tudjuk. Néhány berendezést pl. bekapcsolhatunk off-line üzembe, amely ezáltal (átmenetileg) nem használható. Mindazonáltal, a berendezéslisták módosítása kockázatos vállalkozás (l. a

25-ös megszakítást, azon megjegyzésekre vonatkozóan, hogyan lehet a berendezéslistákat manipulálni és megbízható eredményt kapni.)

A berendezéslista formátumát az eredeti PC típusra határozták meg. Ennek eredményeként a lista bizonyos részei furcsa módon mégsem illeszkednek más típusokhoz. A PCjr-t megelőző összes IBM-modellen a 13. bit (a soros nyomtató) nincs értelmezve és nem is használják. JR A Junioron beállításra kerül, valahányszor a ROM-BIOS-ban levő indítórutin nem talál párhuzamos nyomtatót, de talál a soros portra megfelelően csatlakoztatott berendezést.

A 18-as megszakítás (hexa 12): a memória méretének jelzése

A 18-as megszakítás révén érhetjük el azt a szolgáltatást, amely kbájtokban jelzi a rendelkezésünkre álló memória méretét. Ez pontosan megegyezik a hexa 413 memóriacímen tárolt információval (☛ l. 3. fejezet). Az értéket az *AX*-ben jelzi (l. a 17. információt kiegészítő információt). A szabvány PC-típusokban ez a gép kapcsolóin beállított értékek alapján áll elő. Ezeket a kapcsolókat úgy kell beállítani, hogy azok a tényleges memóriát tükrözzék, bár bizonyos körülmények között a ténylegesen rendelkezésre állónál kisebb értékűre is állíthatjuk.

JR  A PCjr-ben a memóriaméretet a bekapcsoláskor lefutó program határozza meg. A Junior a jelzett memóriaméretet 16K-val módosítja, hogy a képernyő számára memóriát biztosítson. Amikor egy program a 9-es és a 10-es videoüzemmódban működik, a képernyő 32K-t használ, ez csökkenti a felhasználható memóriát. Ez azonban a ROM-BIOS által feljegyzett memóriaméretben nem tükröződik. A 9-es és a 10-es videoüzemmódok nem tartós állapotok, csak egy programon belül lehet őket használni. Ezért nem kell ezekben az üzemmódokban a memória méretére vonatkozó információt megváltoztatni.

A 24-es megszakítás (hexa 18): a BASIC betöltése

A 24-es megszakítást a legtöbbször a ROM-BASIC aktiválására használjuk. Ez a megszakítás szolgáltatásként teszi lehetővé az alap-BASIC felülbírálását. JR A PCjr ezzel a megoldással biztosítja, hogy bekapcsoláskor a ROM-ban lévő BASIC helyett a kazettás (cartridge) BASIC jelentkezzen be.

A 24-es megszakítással bármely program aktiválhatja a BASIC-et (vagy bármi mást, amely azt helyettesítette). Ennek célja lehet a BASIC elindítása is és egy ellenőrizhetetlen program leállítása is. (Igaz, az utóbbira jobb a 25-ös megszakítás.)

A 25-ös megszakítás (hexa 19): az indító (bootstrap) betöltése

A 25-ös megszakítás a számítógép számára a szabvány rendszerbetöltő rutint, ez a bekapcsoláshoz hasonló, és a *Ctrl-Alt-Del* billentyűkombináció hatásával közel azonos eredményt ad. Elkerüli azonban a bekapcsolási rutin hosszú memóriaellenőrzését, és a *Ctrl-Alt-Del* alaphelyzetbe hozási műveleteit.

Ennek a megszakítási szolgáltatásnak két felhasználásáról tudunk. Az egyik a program azonnali leállítása („lelövése"). Akkor tehetjük ezt a programmal, ha az olyan helyzettel találja szembe magát, amelyet megengedhetetlennek tart (pl. a másolásvédelem nyilvánvaló megsértését). Számos másolásvédett program éppen ezért így fejezi be futását, ha valamilyen gyanús jelenséget vesz észre. (Egyes védett programok azért is így tesznek, mert a rendszerbetöltés a memóriát is felülírja. Ha a memóriát nem hoznák alaphelyzetbe, akkor a védett program memóriában lévő része egy hibakereső programmal felderíthető lenne. A lektor.)

A művelet másik felhasználása a számítógép újraindítása az alaphelyzetbe hozás műveletei nélkül, tehát nem szükséges a memóriaméret újraszámolása és a berendezések listájának ismételt összeállítása. Különösen hasznos ez a megszakítás az olyan program számára, amely e két tételt is módosítja. Az érvelés egyszerű: ha a berendezéslistát vagy a memóriaméretet módosítani akarjuk (pl. bizonyos mennyiségű memóriát fenntartunk a lemezkezelés számára), nem számíthatunk megbízhatóan az összes programra, a DOS-ra sem, hogy ellenőrzik majd a tényleges memóriát vagy a berendezések listáját. De egy program félretehet bizonyos memóriát, megváltoztathatja a memóriaméretet, és így használhatja ezt a megszakítást a rendszer újratöltésére. A DOS aktiválásakor a saját memóriaméretre vonatkozó információját használja, de azt előzőleg módosítottuk. Így elérhetjük, hogy sem a DOS, sem más fejlettebb program nem fog az általunk fenntartott memóriához nyúlni. (A fejlettebb program itt olyan programot jelöl, amely közvetlenül nem használ memóriacímeket, hanem a memória kiosztását a DOS-ra bízza. A lektor.)

Példaként bemutatunk egy kódtöredéket, amely megváltoztatja a memóriaméret BIOS szerinti értékét, és a 25-ös megszakítást használja a számítógép újraindítására.

```
MOV AX,40H                  ; a hexa 40 BIOS adatszegmenst állítja be...
MOV ES,AX                   ; ... az ES szegmens regiszterbe
MOV WORD PTR ES:19,256      ; a memóriaméretet 256K-ra állítja
INT 25                      ; újraindítja a rendszert
```

A 26-os megszakítás (hexa 1A): Időszolgáltatások

A 26-os megszakítás nyújtja az időszolgáltatásokat. Az ebben a fejezetben tárgyalt egyéb megszakításoktól ugyan eltérően, de az összes egyéb ROM-BIOS szolgáltatásokhoz hasonlóan, ezzel a megszakítással egynél több szolgáltatást aktiválhatunk. A két alapszolgáltatást, amelyet 0-val és 1-gyel számozunk, az *AH* regiszterben lehet kiválasztani (l. a 12.12. táblázatot).

A ROM-BIOS kezeli az órát. Ez a rendszeróra ütemeinek számolásán alapul. A számlálást éjféltől kezdi. Minden egyes óraütés, a ROM-BIOS 8-as megszakítás szolgáltatásának rutinja, az óraszámláló értékét eggyel növeli.

Amikor az óraszámláló túlhaladja a 24 órát, a számláló visszaáll 0-ra és jelzi, hogy elmúlt éjfél. A jelzés azonban nem számlálás, tehát nem tudjuk kideríteni, hogy hány éjfél múlt el.

Az óra majdnem pontosan 1 193 180 : 64K ütemmel, vagyis másodpercenként átlag 18,2-szer üt. A számlálót egy 4 bites integerként a hexa 46C memóriahelyen tárolja. Az éjfél számlálóértéket, amelyet az óraszámláló növekedésének összehasonlítására használunk, az 1 573 040 (hexa 1800B0) és a hexa 470 helyen tároljuk (☛ l. a 3. fejezetet). Amikor a DOS-nak az idő ismeretére van szüksége, az időszolgáltatással kiolvassa azt az óraszámlálóból, és ebből a nyers számból számítja ki az időt. Ha az időszámláló érzékeli értéke alaphelyzetbe állítását, akkor növeli a dátumot.

12.12. táblázat
A 26. megszakítás által előidézett két ROM-BIOS-időszolgáltatás és azok regiszterbeállításai

Szolgáltatás	Leírás	Regiszter beállításai
0	Az aktuális óraszámláló kiolvasása	CX = az óraszámláló magasabb helyértékű része DX = az óraszámláló alacsonyabb helyértékű része AL = 0, ha az időmérő nem haladta meg a 24-órás periódust AL < > 0, ha az időmérő új napot számol
1	Az aktuális óraszámláló beállítása	CX = az óraszámláló magas helyértékű része DX = az óraszámláló alacsony helyértékű része

Az alábbi BASIC képletekkel az óraszámlálóból mi is bármikor kiszámíthatjuk az időt.

```
HOUR = CLOCK \ 65543                    (hexa 10007)
REMAINDER = CLOCK MOD 65543
MINUTES = REMAINDER \ 1092              (hexa 444)
REMAINDER = REMAINDER MOD 1092
SECONDS = REMAINDER \ 18.21             A pontos eredményhez 18,21, egyéb-
                                        ként használható osztóként 18 is
REMAINDER = REMAINDER MOD 18.21
HUNDREDTHS = CINT (REMAINDER*100)
```

A fordított úton közel pontos óraszámlálót számolhatunk az időből a következő formulával:

```
COUNT = (HOUR * 65543.33) + (MINUTES * 1092.38)
 + (SECONDS * 18.21) + (HUNDREDTHS * .182)
```

AT Hamarosan látni fogjuk, hogy az AT-vel adott VIOS bővítés tartalmazza azokat az idő- és dátumszolgáltatásokat, amelyek ezeket a feladatokat önműködően végrehajtják.

A 0-s szolgáltatás: az óraszámláló aktuális értékének kiolvasása

A 0-s szolgáltatás az aktuális óraszámlálót helyezi két regiszterbe: a magasabb helyiértékű részt a *CX*-be, az alacsonyabb helyiértékűt pedig a *DX*-be. Az $AL = 0$, ha az utolsó óraérték kiolvasás vagy beállítás óta nem múlt el éjfél, és $AL = 1$, ha elmúlt. Az éjféljel mindig alaphelyzetbe áll, valahányszor az órát kiolvassák. A szolgáltatást használó program felelős azért, hogy az éjféljelet nyilvántartsa, és a dátumváltozásokat nyomon kövesse. Általában a DOS-progra-

moknak közvetlenül nem szabad használni ezt a szolgáltatást. Ha mégis használják, akkor vállalkozniuk kell az új dátum kiszámításának és beállításának aprólékos munkájára is.

🕮 Megjegyzés: Érdekes módon a DOS 2.00 változata nem frissíti következetesen a dátumot az éjféljelre. A következő DOS-változat, a 2.10-es, és az összes későbbi DOS ezt már megteszi.

Az 1-es szolgáltatás: az aktuális óraszámláló beállítása

Az 1-es szolgáltatás a hexa 46C-nél levő óraszámlálót állítja be. Az értéket a *CX : DX* regiszterpárból veszi. Valahányszor a számláló a 24. órához ér, az éjféljel visszaállítódik.

Az AT időszolgáltatásai

[AT] A 2 – 6-os szolgáltatást ugyancsak a 26-os megszakítás váltja ki. Ezeket a BIOS AT-változatában vezették be. A 2-es, 3-as és 4-es szolgáltatás kiolvassa, ill. beállítja a valós idejű órát, miközben idő- és dátuminformációt is nyújt. Az 5-ös és a 6-os szolgáltatás pedig egy ébresztőt állít be a megszakításhoz a tényleges időtől számított 24 órán belül. (☛ Ezekről a szolgáltatásokról bővebbet a 13.2. táblázat vagy az AT Technical Reference kézikönyv BIOS listája nyújt.)

13 A ROM-BIOS-szolgáltatások összefoglalása

Bevezetés • 198

Rövid összefoglalás • 198

Hosszú összefoglalás • 200

Bevezetés

Ez a fejezet a 8 – 12. fejezetekben tárgyalt ROM-BIOS szolgáltatási rutinok összegzését nyújtja, és gyors referenciaként szolgál. Miután már megértettük a ROM-BIOS szolgáltatásokat, ezekből a táblázatokból mindazt a programozási információt megtudhatjuk, amire szükségünk lehet.

Rövid összefoglalás

Ebben a fejezetben röviden felsoroljuk az összes ROM-BIOS-szolgáltatást, hogy ezeket egy pillantással át lehessen fogni.

13.1. táblázat
A ROM-BIOS szolgáltatásainak rövid összefoglalása

Tárgy	Megszakítás		Szolgáltatás, hexa	Leírás	Modell-speci-fikáció
	Dec	Hexa			
Képernyő kinyomtatás	5	5	n/a	A képernyőtartalom nyomtatása	
Video	16	10	0	A videoüzemmód beállítása	
Video	16	10	1	A kurzorméret beállítása	
Video	16	10	2	A kurzorhely beállítása	
Video	16	10	3	A kurzorhely kiolvasása	
Video	16	10	4	A fényceruza helyzetkiolvasása	
Video	16	10	5	Aktív monitoroldal beállítása	
Video	16	10	5 (*AL*:128)	A monitorregiszterek lekérdezése	
Video	16	10	5 (*AL*:129)	A CPU-monitorregiszter beállítása	
Video	16	10	5 (*AL*:130)	A CRT-monitorregiszter beállítása	
Video	16	10	5 (*AL*:131)	Mindkét monitorregiszter beállítása	
Video	16	10	6	Egy ablak görgetése felfelé	
Video	16	10	7	Egy ablak görgetése lefelé	
Video	16	10	8	A karakter és az attribútum kiolvasása	
Video	16	10	9	A karakter és az attribútum kiolvasása	
Video	16	10	A	A karakter írása	
Video	16	10	B	A színskála beállítása	
Video	16	10	C	A képpont (pixel) beírása	
Video	16	10	D	Képpont kiolvasása	
Video	16	10	E	A karakter TTY-kénti írása	
Video	16	10	E	Az aktuális videoüzemmód lekérdezése	
Video	16	10	10 (*AL*:0)	Egy palettaregiszter beállítása	JR
Video	16	10	10 (*AL*:1)	Egy szegélyregiszter beállítása	JR
Video	16	10	.10 (*AL*:2)	Az összes palettaregiszter beállítása	JR
Video	16	10	13	Karakterlánc írása	AT
Berendezés	17	11	n/a	A periféria berendezések listájának lekérdezése	
Memória	18	12	n/a	A használható memóriaméret (kbájtban) lekérdezése	

Tárgy	Megszakítás		Szolgáltatás, hexa	Leírás	Modeli-speci-fikáció
	Dec	Hexa			
Lemez	19	13	0	A hajlékonylemezes egység alapállapotba állítása	
Lemez	19	13	1	A hajlékonylemezes egység állapotának lekérdezése	
Lemez	19	13	2	Szektorok kiolvasása a hajlékonylemezről	
Lemez	19	13	3	Szektorok írása a hajlékonylemezre	
Lemez	19	13	4	A hajlékonylemezen lévő szektorok ellenőrzése	
Lemez	19	13	5	Sáv formatálása a hajlékonylemezen	
Lemez	19	13	8	Az aktuális lemezegység paramétereinek olvasása	AT
Lemez	19	13	9	A merevlemezes egység paramétertáblájának inicializálása	AT
Lemez	19	13	A	Hosszú olvasás	AT
Lemez	19	13	B	Hosszú írás	AT
Lemez	19	13	C	Cilinderkeresés	AT
Lemez	19	13	D	A lemezkiválasztás alapállapotba állítása	AT
Lemez	19	13	10	A lemezegység készenlétének ellenőrzése	AT
Lemez	19	13	11	A lemezegység újrakalibrálása	AT
Lemez	19	13	14	A vezérlő diagnosztikája	AT
Lemez	19	13	15	A lemeztípus lekérdezése	AT
Lemez	19	13	16	A lemezállapot megváltoztatása	AT
Lemez	19	13	17	A lemeztípus beállítása	AT
Soros port	20	14	0	A soros port paramétereinek inicializálása	
Soros port	20	14	1	Egy karakter kiküldése	
Soros port	20	14	2	Egy karakter vétele	
Soros port	20	14	3	A soros port állapotának lekérdezése	
Kazetta	21	15	0	A kazettamotor bekapcsolása	
Kazetta	21	15	1	A kazettamotor kikapcsolása	
Kazetta	21	15	2	Adattömbök kiolvasása	
Kazetta	21	15	3	Adattömbök írása	
Készülék	21	15	80	A készülék nyitása	AT
Készülék	21	15	81	A készülék zárása	AT
Készülék	21	15	82	A készülék programjának befejezése	AT
Készülék	21	15	83	Várakozás	AT
Botkormány	21	15	84	Botkormány-támogatás	AT
Rendszerkérdés	21	15	85	A *Sys Req*-billentyű lenyomása	AT
Készülék	21	15	86	Várakozás	AT
Készülék	21	15	87	Tömbmozgatás	AT
Memória	21	15	88	A bővített memória lekérdezése	AT
Memória	21	15	89	Virtuális memória kapcsolás	AT
Készülék	21	15	90	Készülék foglalt hurok	AT
Készülék	21	15	91	A jelző és az összes megszakítás beállítása	AT
Billentyűzet	22	16	0	A következő billentyűkarakter kiolvasása	
Billentyűzet	22	16	1	A karakterkészenlét jelentése	
Billentyűzet	22	16	2	A váltóállapot lekérdezése	

Tárgy	Megszakítás		Szolgáltatás, hexa	Leírás	Modell-specifikáció
	Dec	Hexa			
Billentyűzet	22	16	3 (*AL*:0)	A typematic visszaállítása	JR
Billentyűzet	22	16	3 (*AL*:1)	A kezdeti késleltetés növelése	JR
Billentyűzet	22	16	3 (*AL*:2)	A folytatódó késleltetés növelése	JR
Billentyűzet	22	16	3 (*AL*:3)	Mindkét késleltetés növelése	JR
Billentyűzet	22	16	3 (*AL*:4)	A typematic lekapcsolása	JR
Billentyűzet	22	16	4 (*AL*:0)	Klikk lekapcsolva	JR
Billentyűzet	22	16	4 (*AL*:1)	Klikk bekapcsolva	JR
Nyomtató	23	17	0	Egy bit küldése a nyomtatóhoz	
Nyomtató	23	17	1	A nyomtató inicializálása	
Nyomtató	23	17	2	A nyomtató állapotának lekérdezése	
BASIC	24	18	n/a	A vezérlés átkapcsolása a BASIC-re	
Bootstrap	25	19	n/a	A számítógép újraindítása	
Idő	26	1A	0	Az óraszámláló aktuális értékének kiolvasása	
Idő	26	1A	1	Az óraszámláló aktuális értékének beállítása	
Idő	26	1A	2	A valós idejű óra kiolvasása	AT
Idő	26	1A	3	A valós idejű óra beállítása	AT
Idő	26	1A	4	A dátum kiolvasása a valós idejű órából	
Idő	26	1A	5	A dátum beállítása a valós idejű órán	AT
Idő	26	1A	6	A riasztás beállítása	AT
Idő	26	1A	7	A riasztás visszaállítása	AT

Hosszú összefoglalás

Kiegészítjük az előző táblázatot a paraméterek be- és kivitelére szolgáló rendszerhasználat leírásával. Az előző táblázat leginkább arra használható, hogy gyorsan megtaláljuk, melyik szolgáltatásra van szükségünk. A következő táblázatban az egyes szolgáltatások használatát részletezzük.

13.2. táblázat
A ROM BIOS-szolgáltatások kiegészített listája

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
A képernyő-tartalom kinyomtatása	05	*AH* = 05	n/a	A képernyő-tartalmat a nyomtatóra küldi. Az állapot- és eredménybájt alsó memóriacíme, hexa 500-on (0050 : 0000)

Szolgáltatás	Megszakítás, hexa	Regiszter: input	Regiszter: output	Megjegyzés
Videoszolgáltatások				
A videoüzemmód beállítása	10	*AH* = 00 *AL* = videoüzemmód	nincs	Videoüzemmód az *AL*-ben: 00 : 40 × 25 szöveg, 16 szürke 01 : 40 × 25 szöveg, 16 vagy 8 szín 02 : 80 × 25 szöveg, 16 szürke 03 : 80 × 25 szöveg, 16 vagy 8 szín 04 : 320 × 200 grafika, 4 szín 05 : 320 × 200 grafika, 4 szürke 06 : 640 × 200 grafika, t/f 07 : 80 × 25 szöveg, t/f 08 : 160 × 200 grafika, 16 szín 09 : 320 × 200 grafika, 16 szín 0A : 640 × 200 grafika, 4 szín
A kurzorméret beállítása	10	*AH* = 01 *CH* = a sorletapogatás kezdése *CL* = a sorletapogatás befejezése	nincs	A színes–grafikus adapter a 0–7. sort használja, a monokrom adapter pedig a 0–13. sort
A kurzorhelyzet beállítása	10	*AH* = 02 *BH* = a képernyőoldal száma *DH* = sor *DL* = oszlop	nincs	
A kurzorhelyzet kiolvasása	10	*AH* = 03 *BH* = a képernyőoldal száma	*CH* = a sorlepatogatás kezdése *CL* = a sorletapogatás befejezése *DH* = sor *DL* = oszlop	

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
A fényceruza helyzetének kiolvasása	10	*AH* = 4	*AH* = a ceruzatrigger jele *BX* = a képpont oszlopa *CH* = a képpont sora *DH* = a karakter sora *DL* = a karakter oszlopa	
Az aktív képernyőoldal beállítása	10	*AH* = 05 *AL* = oldalszám		
A képernyőoldal regiszterének lekérdezése	10	*AH* = 05 *AL* = 80	*BH* = a CRT-oldal regisztere *BL* = a CPU-oldal regisztere	
A CPU-képernyőoldal regiszterének beállítása	10	*AH* = 05 *AL* = 81 *BL* = a CPU-oldal regisztere	*BH* = a CRT-oldal regisztere *BL* = a CPU-oldal regisztere	
A CRT-képernyőoldal regiszterének beállítása	10	*AH* = 05 *AL* = 82 *AB* = a CRT-oldal regisztere	*BH* = a CRT-oldal regisztere *BL* = a CPU-oldal regisztere	
Mindkét képernyőoldal regiszterének beállítása	10	*AH* = 05 *AL* = 83 *BH* = a CRT-oldal regisztere	*BH* = a CRT-oldal regisztere *BL* = a CPU-oldal regisztere	
Egy ablak görgetése felfelé	10	*AH* = 06 *AL* = a felfelé mozgatás sorának száma *BH* = kitöltő-attribútum *CH* = felső sor *CL* = bal oszlop *DH* = alsó sor *DL* = jobb oszlop	nincs	

Szolgáltatás	Meg-szakí-tás, hexa	Regiszter		Megjegyzés
		input	output	
Egy ablak görgetése lefelé	10	*AH* = 07 *AL* = a lefelé mozgatás sorának száma *BH* = kitöltő attribútum *CH* = felső sor *CL* = bal oszlop *CH* = alsó sor *DL* = jobb oszlop	nincs	
Karakter és attribútum kiolvasása	10	*AH* = 08 *BH* = a képernyőoldal száma	*AH* = karakter *AL* = attribútum	
Karakter és attribútum beírása	10	*AH* = 09 *AL* = karakter *BH* = oldalszám *BL* = attribútum *CX* = az ismétlődő karakterek száma	nincs	
Karakter írása	10	*AH* = 0A *AL* = karakter *BH* = oldalszám *BL* = színgrafikus üzemmód *CX* = a karakter-számláló	nincs	
A színpaletta beállítása	10	*AH* = 0B *BH* = a palettaszín azonosítója *BL* = a palettával használt kiegészítő szín	nincs	
Képpont írása	10	*AH* = 0C *AL* = szín *CX* = a képpont oszlopa *DL* = a képpont sora	nincs	
Képpont olvasása	10	*AH* = 0D *CX* = a képpont oszlopa *DL* = a képpont sora	*AL* = színkiolvasás	

Szolgáltatás	Meg-szakí-tás, hexa	Regiszter		Megjegyzés
		input	output	
Karakter TTY-ként írása	10	*AH* = 0E *AL* = karakter *BL* = szín a grafikus nyomtatóhoz	nincs	
Az aktuális videoüzemmód lekérdezése	10	*AH* = 0F	*AH* = szélesség (karakterekben) *AL* = videoüzemmód *BH* = oldalszám	
[JR] Egy paletta-regiszter beállítása	10	*AH* = 10 *AL* = 00 *BH* = palettaérték *BL* = palettaregiszter	nincs	
[JR] A szegélyregiszter beállítása	10	*AH* = 10 *AL* = 01 *BH* = szegélyszín	nincs	
[JR] Az összes palettaregiszter beállítása	10	*AH* = 10 *AL* = 02 *ES* : *DX* = palettaérték-mutató	nincs	
[AT] Karaktersorozat írása kurzormozgatás nélkül	10	*AH* = 13 *AL* = 00 *BL* = attribútum *BH* = a képernyő-oldal száma *DX* = a kurzor kezdőhelyzete *CX* = a karakterek száma *ES* : *BP* = az első karakter		
[AT] Karaktersorozat írása a kurzor mozgatásával	10	*AH* = 13 *AL* = 01 *BL* = attribútum *BH* = a képernyő-oldal száma *DX* = a kurzor kezdőhelyzete *CX* = a karakterek száma *ES* : *BP* = az első karakter	nincs	

Szolgáltatás	Meg-szakí-tás, hexa	Regiszter		Megjegyzés
		input	output	
AT Váltakozó karakterek és attribútumok írása kurzormozgatás nélkül	10	*AH* = 13 *AL* = 02 *BH* = a képernyő-oldal száma *DX* = a kurzor kezdőhelyzete *CX* = a karakterek száma *ES : BP* = az első karakter	nincs	
AT Váltakozó karakterek és attribútumok írása a kurzor mozgatásával	10	*AH* = 13 *AL* = 03 *BH* = a képernyő-oldal száma *DX* = a kurzor kezdőhelyzete *CX* = a karakterek száma *ES : BP* = az első karakter	nincs	
Berendezéslista szolgáltatás				
A kapcsolt perifériák lekérdezése	10	nincs	*AX* = bitkód berendezéslista	Bitbeállítás az *AX*-ben: 00 = lemez-meghajtó 01 = matematikai koprocesszor 02, 03 = rendszer-kártyán levő RAM (16K-s blokkokban) 04, 05 = kezdeti videoüzemmód 00 = nem használt 01 = 40 × 25 színes 10 = 80 × 25 színes 11 = 80 × 25 f/f 06, 07 = a lemezmeghajtók száma 08 = van-e DMA? 00 = igen; 01 = nem 09, 10, 11 = RS – 232 kártyák száma a rendszerben

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
A kapcsolt perifériák lekérdezése	10	nincs	AX = bitkód berendezéslista	12 = játék I/O csatolva (AT nem használt) 13 = soros nyomtató csatlakoztatva 14, 15 = a csatlakoztatott nyomtató száma
Memóriaszolgáltatások				
Használható memóriaméret (kbájtban)	12	nincs	AX = memóriaméret	
Hajlékonylemezes szolgáltatások				
A hajlékonylemezes rendszer visszaállítása	13	AH = 00	nincs	
A hajlékonylemez állapotának lekérdezése	13	AH = 01	AL = állapotkód	Állapotkódok $AL = 1$: hibás utasítás $AL = 2$: címjelzést nem talált $AL = 3$: írást kísérelt meg írásvédett lemezen $AL = 4$: szektort nem talált $AL = 6$: a lemez ki van véve $AL = 8$: DMA-túlfutás $AL = 9$: DMA túl van a 64K határon

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
A hajlékonylemez állapotának lekérdezése	13	*AH* = 01	*AL* = állapotkód	*AL* = 10: rossz CRC *AL* = 20: a NEC-vezérlő hibás *AL* = 40: a keresés eredménytelen *AL* = 80: a várakozási-idő-hiba
A hajlékonylemezes szektorok olvasása	13	*AH* = 02 *AL* = szektorok száma *CH* = sávszám *CL* = szektorszám *DH* = fejszám *DL* = meghajtószám *ES* : *BX* = puffermutató	*CH* = siker vagy kudarc *AH* = állapotkód *AL* = az olvasott szektorok száma	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
A hajlékonylemezes szektorok írása	13	*AH* = 03 *AL* = szektorok száma *CH* = sávszám *CL* = szektorszám *DH* = fejszám *DL* = meghajtószám *ES* : *BX* = puffermutató	*CF* = siker vagy kudarc jelzése *AH* = állapotkód *AL* = az írt szektorok száma	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
A hajlékonylemezes szektorok ellenőrzése	13	*AH* = 04 *AL* = szektorok száma *CH* = sávszám *DL* = szektorszám *DH* = fejszám *DL* = meghajtószám	*CF* = siker vagy kudarc jelzése *AH* = állapotkód *AL* = az ellenőrzött szektorok száma	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
Sáv formátolása hajlékonylemezen	13	*AH* = 05 *AL* = szektorok száma *CH* = sávszám *CL* = szektorszám *DH* = fejszám *DL* = meghajtószám *ES* : *BX* = a 4 bájtos címmezők listamutatója	*CF* = a siker vagy kudarc jelzése *AH* = az állapotkód	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
Sáv formátolása hajlékony-lemezen	13	2. bájt-fej 3. bájt-szektor 4. bájt = bájt/szektor	1.	bájtsáv
AT Az aktuális meghajtó paraméterének lekérdezése	13	*AH* = 08	*DL* = a meghajtószám *DH* = az oldalak max. száma *CL* = a szektorok max. száma *CH* = a sávok max. száma *CF* = siker vagy kudarc jelzése *AH* = az állapotkód	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
AT Két Winchester-alaptábla inicializálása	13	*AH* = 09	*CF* = siker vagy kudarc jelzése *AH* = állapotkód	A 41-es megszakítás a 0. meghajtótáblára mutat A 46-os megszakítás az 1. meghajtótáblára mutat Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
AT Hosszú olvasás	13	*AH* = 0A *DL* = meghajtó-azonosító *DH* = fejszám *CH* = cilinderszám *CL* = szektorszám *ES* : *BX* = puffermutató	*CF* = siker vagy kudarc jelzése *AH* = állapotkód	Az állapot-kódokat l. az *AH* = 01-es szolgáltatásnál!
AT Hosszú írás	13	*AH* = 0B *DL* = meghajtó-azonosító *DH* = fejszám *CH* = szektorszám *ES* : *BX* = puffermutató	*CF* = siker vagy kudarc jelzése *AH* = állapotkód	Az állapot-kódokat l. az *AH* = 01-es szolgáltatásnál!
AT Cilinderkeresés	13	*AH* = 0C *DL* = meghajtó-azonosító *DH* = fejszám *CH* = szektorszám	*CF* = siker vagy kudarc jelzése *AH* = állapotkód	Az állapot-kódokat l. az *AH* = 01-es szolgáltatásnál!

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
AT Váltakozó lemez visszaállítása	13	*AH* = 0D *DL* = meghajtó-azonosító	*CF* = siker vagy kudarc jelzése *AH* = állapotkód	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
AT A meghajtó készenlétének ellenőrzése	13	*AH* = 10 *DL* = meghajtó-azonosító	*CF* = siker vagy kudarc jelzése *AH* = állapotkód	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
AT A meghajtó újrakalibrálása	13	*AH* = 11 *DL* = meghajtó-azonosító	*CF* = siker vagy kudarc jelzése *AH* = állapotkód	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
AT A vezérlő diagnosztikája	13	*AH* = 14	*CF* = siker vagy kudarc jelzése *AH* = állapotkód	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
AT A lemez típusának lekérdezése	13	*AH* = 15 *DL* = meghajtó-azonosító	*AH* = lemeztípus *CX, DX* = 512 bájtos szektorok száma, ha *AH* = 3	Lemeztípusok: *AH* = 0: nincs *AH* = 1: változás nincs *AH* = 2: változás *AH* = 3: Winchester
AT A lemez állapotának változtatása	13	*AH* = 16	*DL* = meghajtó, melynek lemezcseréje volt *AH* = lemezcsere állapot 00 = nincs lemezcsere 01 = lemezcsere volt	
AT A lemez típusának beállítása	13	*AH* = 17 *AL* = lemeztípus		Lemeztípusok az AH-ban: *AL* = 00: nincs lemez *AL* = 01: szokásos lemez a szokásos meghajtóban *AL* = 03: nagy kapacitású (1, 2 Mbyte) lemez a nagy kapacitású tárolóban

Szolgáltatás	Meg-szakí-tás, hexa	Regiszter		Megjegyzés
		input	output	
A soros port szolgáltatásai				
A soros port paramétereinek inicializálása	14	*AH* = 00 *DX* = a soros port száma	*AX* = a soros port állapota	Az állapotbit beállítása: 00, 01 = szóhossz: 10 = 7 bit, 11 = 8 bit 02 = stop bitek, 0 = 1; 1 = 2 03, 04 = paritás 00, 01 = nincs; 01 = páratlan 11 = páros 05, 06, 07 = átviteli sebesség; 000 = 110; 001 = 150; 010 = 360; 011 = 600; 100 = 1200; 101 = 2400; 110 = 4800; 111 = 9600 (4800 a PCjr-on)
Egy karakter kiküldése	14	*AH* = 01 *AL* = karakter *DX* = a soros port száma	*AH* = siker vagy kudarc jelzése *AL* = modemállapot	Az AH bit beállítása: 00 = adat-készenlét; 01 = túlfutáshiba 02 = paritáshiba 03 = kerethiba 04 = a meg-szakítás észlelve 05 = az átviteli pufferregisz-ter üres 06 = az átviteli váltóregiszter üres 07 = várakozási idő

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
Egy karakter kiküldése	14	*AH* = 01 *AL* = karakter *DX* = a soros port száma	*AH* = siker vagy kudarc jelzése *AL* = modemállapot	Az AH bitbeállítása: 00 = a delta törlése átvitelhez 01 = a delta adatkészlete készen 02 = a követőél gyűrűjének érzékelése 03 = változás a vevővonalon 04 = törlés átvitelhez 05 = adatkészlet készen 06 = gyűrű érzékelve 07 = a vételivonal jele
Egy karakter vétele	14	*AH* = 02 *DX* = a soros port száma	*AH* = siker- vagy kudarcjelzés állapotkódja *AL* = karakter	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
A soros port lekérdezése	14	*AH* = 03	*AX* = állapotkód	Az állapotkódokat l. az *AH* = 01-es szolgáltatásnál!
Kazettás szolgáltatások				
A kazettamotor bekapcsolása	15	*AH* = 00	nincs	
A kazettamotor kikapcsolása	15	*AH* = 01	nincs	
Adattömbök kiolvasása	15	*AH* = 02 *CX* = a bájtok száma *ES*:*BX* = adatterület mutató	*CF* = hibajel *DX* = az olvasott bájtok száma *ES*:*BX* = az utolsó kiolvasott bájt mögé mutató	
Adattömbök írása	15	*AH* = 03 *CX* = az írt bájtok száma *ES*:*BX* = adatterület-mutató	*ES*:*BX* = az utolsó bájt mögé mutató	

Szolgáltatás	Megszakítás, hexa	Regiszter input	Regiszter output	Megjegyzés
Az AT bővített szolgáltatásai				
AT A készülék nyitva	15	*AH* = 80 *BX* = készülék-azonosító *CX* = a folyamat típusa	nincs	
AT A készülék zárva	15	*AH* = 81 *BX* = készülék-azonosító *CX* = a folyamat típusa	nincs	
AT A készülék-program leállt	15	*AH* = 82 *BX* = készülék-azonosító	nincs	
AT Várakozás	15	*AH* = 83 *AL* = alszolgáltatás: 0 = időszak-beállítás 1 = törlés *ES*:*BX* = a hívó-memória-mutató *CX*, *DX* = a várakozás ideje (μsec)	nincs	
AT Botkormány-támogatás	15	*AH* = 84 *DX* = 0 az aktuális kapcsolóbeállítás lekérdezése	*AL* = kapcsoló-beállítás	
AT Botkormány-támogatás	15	*AH* = 84 *DX* = 1 bemenet kiolvasása	*AX* = A(x) érték *BX* = A(y) érték *CX* = B(x) érték *DX* = B(y) érték	
AT A rendszerindító billentyű lenyomása	15	*AH* = 85 *AL* = 00 lenyomva *AL* = 01 megszakítva	nincs	
AT Várakozás	15	*AH* = 86 *CX*, *DX* = a visszatérés előtti várakozás ideje (μsec)	nincs	

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
AT Tömbmozgatás	15	*AH* = 87 *CX* = a mozgatandó szavak száma *ES* : *SI* = táblamutató	nincs	
AT A memória-bővítés lekérdezése	15	*AH* = 88	nincs	
AT Virtuális üzemmódba kapcsolás	15	*AH* = 89	nincs	Figyelmeztetés: használata előtt l. a BIOS listázást!
AT A készülék foglaltsági hurok	15	*AH* = 90	nincs	L. a BIOS listázást!
AT A zászló beállítása és a megszakítás elvégzése	15	*AH* = 91 *AL* = típuskód	nincs	L. a BIOS listázást!
Billentyűzetszolgáltatások				
A következő billentyűzet-karakter kiolvasása	16	*AH* = 00	*AH* = letapogatókód (kiegészítőbájt) *AL* = karakterkód (főbájt)	
A karakter készenlét jelzése	16	*AH* = 01	*ZF* = készen vagy sem jelzés *AH* = letapogatókód (kiegészítőbájt) *AL* = karakterkód (főbájt)	
A váltó állapotának lekérdezése	16	*AH* = 02	*AL* = váltóállapot-bitek	Váltóállapot-bitek: 0. bit = 1: jobb váltó lenyomva 1. bit = 1: bal váltó lenyomva 2. bit = 1: *Ctrl* lenyomva 3. bit = 1: *Alt* lenyomva

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
A váltó állapotának lekérdezése	16	*AH* = 02	*AL* = váltóállapot-bitek	4. bit = 1: *Scroll Lock* aktív 5. bit = 1: *Num Lock* aktív 6. bit = 1: *Caps Lock* aktív 7. bit = 1: beszúrás aktív
JR A typematic visszaállítása	16	*AH* = 03 *AL* = 00	nincs	
JR A kezdeti várakozás növelése	16	*AH* = 03 *AL* = 01	nincs	
JR A folytatólagos várakozás növelése	16	*AH* = 03 *AL* = 02	nincs	
JR Mindkét várakozás növelése	16	*AH* = 03 *AL* = 03	nincs	
JR A typematic lekérdezése	16	*AH* = 03 *AL* = 04	nincs	
JR A billentyűhang lekapcsolása	16	*AH* = 04 *AL* = 00	nincs	
JR A billentyűhang bekapcsolása	16	*AH* = 04 *AL* = 01	nincs	
Nyomtatószolgáltatások				
Egy bájt nyomtatóra küldése	17	*AH* = 00 *AL* = karakter	*AH* = siker vagy kudarc jelzése állapotkód	Állapotbitek: 0. = várakozási hiba 1. = nem használt 2. = nem használt 3. = I/0 hiba 4. = 1: választva 5. = 1: papírhiány 6. = 1: igazolás 7. = nem foglalt

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
A nyomtató inicializálása	17	*AH* = 01	*AH* = állapotkód	Az állapotkódokat l. a 00ás nyomtató-szolgáltatásnál!
Nyomtatóállapot lekérdezése	17	*AH* = 02	*AH* = állapotkód	Az állapotkódokat l. a 00ás nyomtató-szolgáltatásnál!
Egyéb szolgáltatások				
A vezérlés átadása a BASIC-nek	18	nincs	n/a	Nincs visszatérés és output
A komputer újraindítása	19	nincs	n/a	Nincs visszatérés és output
Időszolgáltatások				
Az aktuális óraszámláló kiolvasása	1A	*AH* = 00	*AL* = éjféljel *CX* = az ütemszámláló felső része *DX* = az ütemszámláló alsó része	
Az aktuális óraszámláló beállítása	1A	*AH* = 01 *CX* = az ütemszámláló felső része *DX* = az ütemszámláló alsó része	nincs	
[AT] A valós idejű óra kiolvasása	1A	*AH* = 02	*CH* = órák *CL* = percek *DH* = másodpercek	
A valós idejű óra beállítása	1A	*AH* = 03 *CH* = órák *CL* = percek *DH* = másodpercek *DL* = 1, ha nyári időszámítás 0, ha rendes időszámítás		

Szolgáltatás	Megszakítás, hexa	Regiszter		Megjegyzés
		input	output	
A dátum kiolvasása a valós idejű órából	1A	*AH* = 04	*DL* = nap *DH* = hónap *CL* = év *CH* = évszázad (19 vagy 20)	
A dátum beállítása a valós idejű órán	1A	*AH* = 05 *DL* = nap *DH* = hónap *CL* = év *CH* = évszázad (19 vagy 20)		
A riasztás beállítása	1A	*AH* = 06 *CH* = órák *CL* = percek *DH* = másodpercek		A riasztórutin címét feltételezi a 4A megszakítás helyén
A riasztás beállítása	1A	*AH* = 07		

14 A DOS alapjai

Bevezetés • 218

Érvek és ellenérvek • 218

A DOS: a lemezes szolgáltatások tárháza • 219
A DOS és a video: egy nehéz illesztés • 219

Eltérések a DOS-változatok között • 220

A lemezformátum megválasztásának szempontjai • 221

Megjegyzések • 221

Bevezetés

A 15–18. fejezet a DOS programozást támogató szolgáltatásaira összpontosít. Az utolsó fejezet, ebben a sorozatban a 18., az egyes szolgáltatások műszaki részleteinek összefoglalója. A következőkben bemutatunk néhány olyan szempontot, amelyeket a programozónak figyelembe kell vennie, amikor a DOS szolgáltatásait igénybe veszi. A DOS-szolgáltatások elnevezést a DOS által programjaink számára nyújtott teljes műveleti készlet meghatározására használjuk. Ez az elnevezés a DOS-kézikönyvben nem szerepel. A DOS terminológiájában ezek a szolgáltatások két kategóriába tartoznak: DOS-megszakítások és DOS-függvényhívások. Ezt az elkülönítést nem tervezték el előre, csupán abból a szándékból eredt, hogy a DOS elődjével (a CP/M operációs rendszerrel) ésszerű mértékű kompatibilitást érjenek el.

A DOS-megszakításokat az **INT** utasítással és az egyedi megszakításkódokkal aktiváljuk. A DOS-függvényhívásokat viszont a ROM-BIOS szolgáltatásokhoz hasonlóan, a 33-as megszakítás (hexa 21) révén lehet elérni. Ugyanúgy, mint a ROM-BIOS szolgáltatásoknál, az egyedi függvényeket az *AH* regiszterrel választjuk ki.

A programozás és a tervezés szempontjából a függvényhívás mechanizmusa lényegesen hatékonyabb, mint az egyedi megszakítások csoportja. Fő előnye, hogy korlátlan számú új szolgáltatás hozzáadását teszi lehetővé, mivel minden egyes szolgáltatást egyetlen egyedi megszakításon keresztül hív. A DOS 2.00 változattal bevezetett szolgáltatások mindegyike a függvényhívásokat, nem pedig a megszakításokat bővíti. A DOS 3. változatok szolgáltatásainak többsége is függvényhívás, csak egy új megszakítás van.

Érvek és ellenérvek

Fejlettebb programok tervezésénél és fejlesztésénél természetesen felmerülhet az a kérdés, vajon használjuk-e a DOS-szolgáltatásokat vagy ne. Általános érvényű tanácsunkat e könyvben többször is megismételjük: mindig azt a lehető legmagasabb szintű szolgáltatást használjuk, amely igényeinknek legjobban megfelel. Ez azt jelenti, hogy elsőként a programnyelv beépített szolgáltatásait kell használnunk (ahol ez lehetséges); csak akkor forduljunk a DOS- vagy a ROM-BIOS-szolgáltatásokhoz, amikor erre okvetlenül szükségünk van. A számítógép hardver közvetlen programozásához csak akkor fogjunk hozzá, ha minden kötél szakad.

A gyakorlatban egy programot vagy teljes egészében megírhatunk a programozási nyelv lehetőségeinek korlátai között, vagy csaknem az összes I/O munkát a programozási nyelven kívül, alacsonyabb szinten kell elvégeznünk. Amikor alacsonyabb szintű programozásra van szükség – néhány kivételtől eltekintve –, a DOS-szolgáltatások a legalkalmasabbak a lemezes műveletekre. Ha a billentyűzettel vagy egyéb I/O berendezéssel dolgozunk, elegendők a DOS-rutinok vagy a ROM-BIOS-rutinok – a felhasználástól függetlenül. Sokkal összetettebb a helyzet néhány alacsonyabb szintű képernyőkezelő programnál. Elfogadható képernyőkimenet csaknem minden esetben ROM-BIOS-szolgáltatást vagy közvetlen hardver programozást igényel, még akkor is, ha néhány esetben legjobb lenne az egészet a DOS-ra bízni. Az utóbbi megállapításról hamarosan részletesebben is szólunk.

A DOS: a lemezes szolgáltatások tárháza

Ha megvizsgáljuk azokat az eszközöket és szolgáltatásokat, amelyeket a programozási nyelvek, a DOS, a ROM-BIOS és a hardver használnak, nyilvánvalóvá válik, hogy a leggazdagabb lemezkezelő szolgáltatásokat a DOS szintjén találjuk. Ez magától értetődik, mivel a DOS (lemezoperációs rendszer lévén) a legerősebb a lemezes műveletek támogatásában.

Mint azt a 16. és 17. fejezetekben részletezzük, azon szolgáltatások többsége, amelyeket a DOS nyújt számunkra a lemezállományok kezelésével áll kapcsolatban. Még az általában programvezérelt szolgáltatás, pl. egy másik program betöltése és végrehajtása (75-ös függvény) is kiválthat lemezes műveleteket. Ebből a szempontból a DOS nem is lemezoperációs rendszer, inkább olyan lemezszolgáltatások rendszere, amelyet a programjaink használatára terveztek. Ha az IBM személyiszámítógép-családra tervezünk programokat, jó ötlet, ha a DOS-t éppen ebből a szemszögből közelítjük: hiszen a DOS nem más, mint a lemezes szolgáltatások tárháza.

A DOS és a video: egy nehéz illesztés

A személyi számítógépeknél a fejlettebb programok általában a képernyőt lehetőleg alacsony szinten kezelik. Gyakran közvetlenül írják a képernyőtárat, ami a legalacsonyabb szint. Más műveleteket (pl. a kurzor mozgatást) általában magasabb szinten, a ROM-BIOS-szolgáltatások segítségével hajtja végre. Kezdetben azért volt erre szükség, mert a DOS nem nyújtott megfelelő videoszolgáltatásokat. De a 2.00 változattól kezdődően lehetővé vált, hogy az igényelt képernyőkezelést az ANSI-meghajtóprogrammal – amelyet ANSI.SYS néven is ismerünk (☛ l. az *A* mellékletet a további részleteket illetően) – kiegészített DOS-szolgáltatások révén végezzük el. Ez a program olyan utasításkészletet használ, amely lefordítás után majdnem mindent végrehajt, amire a képernyő képes. Az ANSI-szolgáltatások mégis kissé nehézkesek, mivel nemcsak azt kívánják meg, hogy a programjaink legalább DOS 2-változattal fussanak, hanem a DOS-t is úgy kell konfigurálni, hogy magába foglalja az ANSI-meghajtót is. Véleményünk szerint a tapasztalatlan számítógép-felhasználókat alaposan megzavarja az ANSI-meghajtó beépítéséhez szükséges eljárás. Ez a tényező a legerősebb érv az ellen, hogy bármilyen ANSI-meghajtót igénylő DOS-lehetőséget használjunk.

Ha csak ezzel a tényezővel állnánk szemben, könnyen juthatnánk arra a következtetésre, hogy a DOS videoszolgáltatásainak használatát teljes egészében kerüljük el. Ez azonban nem ilyen egyszerű. A megjelenő legfejlettebb rendszerkörnyezetek nagy száma, különösen az ablaktechnikát használó rendszereké, feltételezik, hogy a program, amely alattuk fut, használja a hivatalosan rendelkezésre álló operációs rendszer szolgáltatásait, és nem kezeli közvetlenül a hardvert. E környezetek léte is döntő érv amellett, hogy lehetőleg a DOS-szolgáltatásokat használjuk.

A legbölcsebb döntés meghozatalánál sok függ attól, hogy mi a programjaink várható élettartama, és melyik gépcsoportokon tudjuk majd használni. Egy PC-specifikus játékprogramnak az élettartama csupán néhány hónap (ez csaknem az összes játékra jellemző), kevés dolog indokolja, hogy ezek miatt aggódjunk. Más a helyzet az általánosabb üzleti vagy a szakmai felhasználásnál, mert azokat különféle környezetben több évig is használnunk kell. Válasszunk és kockáztassunk!

Eltérések a DOS-változatok között

A DOS 3.10 a DOS hatodik hivatalos változata. Bár tökéletesítése és hibajavítások minden egyes új változatban voltak, a hajtóerő az összes változat esetében a hardver volt; a hardverváltozás rendszeresen lemezegység-változás volt.

A 2.10 és a 3.10 változatok kivételével minden változás a DOS-ban jelentős lemezkezelésbeli módosítást vont maga után (beleértve az új lemezformátumokat). A 2.10 változaton ugyan viszonylag keveset módosítottak, de az a lemezre vonatkozott. A vezérlő fejbeállítási idejét úgy állították be, hogy lehetővé tegye a teljesítményeltérést a PCjr-ben és a hordozható PC-ben használt félmagas meghajtóknál. A 2.10 ugyancsak kijavította azt a néhány jól ismert hibát, amelyet a 2.00 tartalmazott. A 3.10 magába foglalta a hálózati funkciókat, amelyeket a 3.00 változatra terveztek, de nem készültek el kibocsátásának idejére. Az alábbiakban röviden összefoglaljuk a változatok közötti lényeges különbségeket:

1.00 változat: Az egyoldalas, 8 szektoros lemezformátumot támogatja. Az összes alap DOS-szolgáltatást magába foglalja ez a változat.

1.10 változat: A kétoldalas lemez támogatásával bővült, a DOS-szolgáltatások ugyanazok maradtak.

2.00 változat: Megjelent a 9 szektoros egyoldalas és kétoldalas lemez és a Winchester lemez támogatása. A DOS-szolgáltatásait lényegesen kibővítették ebben a változatban (l. ☛ a 17. fejezetet). Kazettatámogatással is kiegészítették a 2.00 változatot, bár ez csak a PCjr bejelentésekor vált ismertté.

2.10 változat: Sem új lemezformátumot, sem új DOS-szolgáltatást nem hozott, a lemezművelet időzítését azonban úgy állította be, hogy az a PCjr-nek és a hordozható PC-nek is előnyére legyen.

3.00 változat: A nagykapacitású hajlékony lemezzel és Winchester formátummal bővült. Megteremtette a hálózati lemez alapjait is.

3.10 változat: A hálózati lemezzel bővült, amelyek magukba foglalják a közös állomány megosztásának lehetőségét.

🕮 MEGJEGYZÉS: A DOS minden változata – néhány részlettől eltekintve – felfelé kompatíbilis az előző változatokkal (ezek a részletek mindig elkerülhetetlennek tűnnek).

14.1. táblázat
A hat DOS-változat és a hardverhez kapcsolódó változtatások

Változat	Dátum	Hardverváltoztatás
1.00	81. VIII. 4.	Eredeti PC-típus (egyoldalas hajlékonylemezes meghajtó)
1.10	82. V. 7.	Kétoldalas hajlékonylemezes meghajtó
2.00	83. III. 8.	XT-típus (Winchester-meghajtó)
2.10	83. X. 20.	PCjr és hordozható PC-típusok (félmagas meghajtók)
3.00	84. VIII. 14.	AT-típus (nagykapacitású hajlékonylemezes meghajtók)
3.10	85. III. 7.	Hálózat (hálózati lemezmeghajtók)

A DOS mindegyik új változata a szoftver fejlesztők között újra és újra felvetette azt a kérdést, melyik DOS-változat legyen a cél, mivel a nagyobb hajlékonylemezes formátumok és a kiterjesztett DOS-szolgáltatások kizárják a korábbi DOS-változatok használatát. Ez a zavaros helyzet a múltban sok programfejlesztőt állított nehéz döntések elé. Hosszú ideig a választás általában a DOS 2 változatára esett. A PC típusok eladása azonban felgyorsult, ezáltal a DOS 1-t használók köre a PC közösségnek egyre kisebb részét teszi ki. Ez elősegíti a DOS 2 változat melletti döntést, és lehetővé teszi az összes szolgáltatásának a használatát. Még a DOS 3 változat megjelenésével is a DOS 2 marad a legjobb választás. De ki tudná megmondani, hogy meddig?

A lemezformátum megválasztásának szempontjai

Nemcsak arról kell döntenünk, melyik DOS-szolgáltatást vagy a DOS melyik változatát használjuk, hanem azt is mérlegelnünk kell, melyik lemezformátumot alkalmazzuk. A PC programokat általában egyoldalas 8 szektoros hajlékonylemezen szállítják, mert ez a formátum a legkisebb közös nevező az összes DOS-formátum számára, ezért bármelyik DOS-változat kezelni tudja. Általános jellegük miatt az egyoldalas 8 szektoros formátumok sokáig hasznosak voltak, ma viszont már nem sok érv szól mellettük. Egyrészt az egyoldalas meghajtók kihaltak a PC-ken, helyüket már hosszú ideje a kétoldalas formátum vette át; másrészt bármely program, amely a DOS 2-t igényli, használni tudja a 9 szektoros formátumot. Valójában ez az a formátum, amelyet a DOS 2.00 és a 2.10 maga is használ. Hogy az IBM kedvébe járjon az utolsó néhány DOS 1 tulajdonosnak, használható a kétoldlas 8 szektoros formátum, de nyugodtan használható a kétoldalas, 9 szektoros formátum is.

A 48-as DOS-függvény hívásával a program majdnem tökéletesen fel tudja fedni, hogy milyen DOS-változattal fut. Ha nem vagyunk biztosak a felhasználói környezetben, be kell építenünk a programunkba egy biztonsági elemet, ez fogja állandóan ellenőrizni, hogy a megfelelő DOS változattal dolgozunk-e. (☛ A további információkat l. a 17. fejezetben.)

Megjegyzések

Általánosságban kevés műszaki információval rendelkezünk a DOS-ról. Sok olyan részlet van, amelyről az IBM és a Microsoft nem tájékoztatja a világot. Sajnálatos módon nem tudunk jelentősen változtatni ezen az információínségen, csupán megpróbálhatjuk továbbadni mindazt, amit felfedeztünk, rámutatva néhány hiányosságra. A következő fejezetekben ezt kíséreljük meg.

🕮 MEGJEGYZÉS: A DOS-szolgáltatások hivatalos információforrása a DOS Technical Reference kézikönyv, amelyet a 2.10 változatra adtak ki. Az előző DOS-változatok ezzel egyenértékű információt a fő DOS-kézikönyv tartalmazza.

15 DOS-megszakítások

Bevezetés • 224

Az öt fő DOS-megszakítás • 224

A 32-es megszakítás (hexa 20): a program befejezése • 225
A 37-es és a 38-as megszakítás (hexa 25 és 26): abszolút lemezolvasás és -írás • 225
A 39-es megszakítási (hexa 27): a program befejezése memóriában maradással • 227
A 47-es megszakítás (hexa 2F): a nyomtatóspooler vezérlése • 228

A három DOS címmegszakítás • 230

A 34-es megszakítás (hexa 22): a befejezési cím • 230
A 35-ös megszakítás (hexa 23): a megszakításkezelő címe • 231
A 36-os megszakítás (hexa 24): a kritikus hibakezelő címe • 231

A programszegmens előtag (PSP) • 234

A PSP belső struktúrája • 235

Programpéldák • 239

Bevezetés

Ebben a fejezetben azokat a DOS szolgáltatásokat tárgyaljuk, amelyek a saját megszakításuk útján érhetők el. (☛ L. a 16. és a 17. fejezetben azokra a DOS-függvényhívásokra vonatkozó információkat, amelyeket a közös megszakítás alatt a függvényszám kiválasztásával hívunk meg.)

Összesen kilenc megszakítás van, amelyeket a 15.1. táblázat sorol fel. Közülük öt a 32-es, a 37–39-es, valamint a 47-es (a hexa 20, a 25–27 és a 2F) megszakítások valódi DOS-megszakítások: mindegyikhez egy-egy meghatározott feladat társul. A többi megszakításnak általánosabb a használata. Talán a legfontosabb a 33-as (hexa 21) megszakítás, amelyet a DOS függvényhívások elérésére használunk (részletesen a 16. és a 17. fejezetben tárgyaljuk). A fennmaradó három megszakítás, a 34–36-os (hexa 22–24) szegmentált címek tárolására való. Ezeket a címeket programjaink úgy állítják be (lehetőség szerint a 37-es DOS-függvényhívást használva), hogy speciális rutinokra mutassanak. Ezután a három címmegszakítás segítségével, ha a feltételek adottak, a DOS lefuttatja az e címen található rutinokat (☛ l. a fejezet címmegszakítással foglalkozó részét).

15.1. táblázat
A kilenc DOS-megszakításszolgáltatás

Megszakítás		Megnevezés
Dec	Hexa	
32	20	A program befejezése: hibátlan lefutást követő befejezés
33	21	Függvényhívás közös megszakítás segítségével
34	22	A befejezési cím
35	23	A megszakítási cím
36	24	A kritikus hibakezelő címe
37	25	Abszolút lemezolvasás
38	26	Abszolút lemezírás
39	27	Fejezd be, de maradj a memóriában!
47	2F	Nyomtatóspool vezérlés (csak a DOS 3 változatnál)

🕮 MEGJEGYZÉS: Az IBM hivatalosan nem helyesli, hogy a programozók a DOS megszakításokat használják. Ezért az IBM változó függvényhívásokat ad a 33-as (hexa 21) megszakítással. Lehetséges, hogy később az új DOS-változatok ezen nem jóváhagyott megszakítások használatát nem fogják támogatni, ezért bölcsebb, ha elkerüljük a használatukat; támaszkodjunk inkább a speciális szolgáltatások DOS függvényhívásaira.

Az öt fő DOS-megszakítás

A kilenc DOS-megszakítás közül öt valódi. Ezalatt azt értjük, hogy hozzájuk beépített megszakításkezelő programok kapcsolódnak, amelyek mindegyike egy meghatározott feladatot hajt végre.

A 32-es megszakítás (hexa 20): a program befezése

A 32-es megszakítást arra használjuk, hogy egy programból kilépve a vezérlést visszaadjuk a DOS-nak. Ez egyenértékű a DOS 0-ás függvénykóddal. (☛ L. a 16. fejezetet.) Ezeket a szolgáltatásokat bármely DOS-változattal használhatjuk a program befejezésére.

A 32-es megszakítás nem zárja le automatikusan az állományokat, tehát kilépés előtt használni kell vagy a 16-os, vagy a 62-es DOS-függvényt a változtatott állományok lezárására. Ha egy állomány változott, de nincs lezárva, az állomány tartalommutatója nem az új hosszat fogja mutatni.

A 34-es, a 35-ös és a 36-os DOS-megszakításokon keresztül a programmal kapcsolatos három címet lehet beállítani. A 32-es megszakítással végrehajtott „takarítás" ezeket a címeket visszaállítja a betöltéskor felvett kezdeti értékükre. A visszaállítás akkor fontos, ha a 32-es megszakítást hívó program egy másik program „gyermekeként" futott. A cél: megvédeni a „szülőt" a gyermeknek szánt rutinoktól. (☛ L. a 75-ös függvényt (hexa 4B) a 17. fejezetben.)

🕮 MEGJEGYZÉS: Amikor a DOS végrehajt egy programot, előállít egy programszegmens előtagot (PSP). A *CS*-regiszterbeli kódszegmenshez képest a nulla eltolási címen található PSP tartalmazza a vezérlőinformációkat, amelyek többek között azt közlik a DOS-szal, hogy hová lépjen a program befejezése után. (☛ A PSP-t részletesen e fejezet végén tárgyaljuk.) A DOS a *CS*-regiszterre támaszkodik azért, hogy a PSP-re mutasson a 32-es, a programbefejezési szolgáltatásainak meghívásakor. A *CS* regiszter változása megzavarhatja ezen szolgáltatás működését.

✱ FIGYELEM: *Ha* FAR-*hívással adjuk át a vezérlést egy szubrutinnak, a* CS-*regiszter megváltozik. Ilyen szubrutin a 32-es megszakítást nem használhatja a program befejezésére.*

A 37-es és a 38-as megszakítás (hexa 25 és 26): abszolút lemezolvasás és -írás

A 37-es és 38-as megszakítást arra használjuk, hogy meghatározott lemezszektorokról olvassunk, ill. a szektorokra írjunk. Csak ezek a DOS-szolgáltatások hagyják figyelmen kívül a lemez logikai struktúráját, és csak egyedi szektorokkal dolgoznak. Nem figyelnek az állományokra az állomány tartalommutatókra és a FAT-ra sem. Az összes többi DOS-szolgáltatás a lemez logikai szerkezetének felhasználásával működik.

A 37-es és a 38-as megszakítás hasonló a megfelelő ROM-BIOS lemezszolgáltatásaihoz, de a szektorok eltérő számozás szerint helyezkednek el. A ROM-BIOS-szolgáltatásoknál a szektorokat háromdimenziós koordinátahelyzetük (sáv vagy cilinder; oldal vagy fej; szektor) szerint választjuk ki: míg a 37-es és a 38-as megszakításnál a szektorokat sorszámmal, az egymást követő szektorok számával azonosítjuk. (☛ A DOS szektorszámozási rendszerét az 5. fejezetben tárgyaltuk.)

A következőkben közöljük azt a BASIC képletet, amely a ROM-BIOS-nál használt háromdimenziós koordinátát a DOS-nál használt szektorsorszámmá alakítja:

```
DOS.SECTOR.NUMBER = (BIOS.SECTOR - 1) + BIOS.SIDE
* SECTORS.PER.TRACK + BIOS.TRACK * SECTORS.PER.TRACK
* SIDES.PER.DISK
```

A következő pedig a szektorsorszámot fordítja háromdimenziós koordinátára:

```
BIOS.SECTOR = 1 + DOS.SECTOR.NUMBER MOD SECTORS.PER.TRACK
```

BIOS.SIDE = (DOS.SECTOR.NUMBER \ SECTORS.PER.TRACK) MOD SIDES.PER.DISK
BIOS.TRACK = DOS.SECTOR.NUMBER \ (SECTORS.PER.TRACK * SIDES.PER.DISK)

MEGJEGYZÉS: A PC leggyakrabban használt lemezformátumára a kétoldalas 9 szektoros lemezre a *SECTORS.PER.TRACK* érték 9 és a *SIDES.PER.DISK*=2. Jegyezzük meg azt is, hogy az oldalakat és a sávokat is másképp számozzák a ROM-BIOS számozási rendszerében: az oldalakat és a sávokat 0-tól, a szektorokat viszont 1-től kezdődően.

A szektor tömbjeinek kiválasztása miatt a szükséges paramétereket külön regiszterekbe tesszük. A szektorok számát a *CX*-regiszterben határozzuk meg, a kezdő szektorszámot a *DX*-regiszterben, az adatátvitelhez a memóriacímet pedig a *DS* : *BS*-ben. A lemezegységet az *AL*-regiszterben választjuk ki. Az *A* egység 0, a *B* pedig 1.

A ROM-BIOS-szolgáltatások valós fizikai lemezegységekkel dolgoznak, a DOS-szolgáltatások viszont, logikaiakkal. A DOS feltételezi, hogy minden számítógépnek legalább két logikai lemezegysége van. Ha nincs *B* fizikai lemezegység, akkor a DOS az egyetlen fizikai lemezegység használatakor vagy az *A*-t, vagy a *B*-t fogja szimulálni, bármelyikre is van szüksége. Ezután ezeket a logikai meghajtókat újra elnevezhetjük a DOS **ASSIGN**-utasításával.

A 37-es és a 38-as megszakítások eredménye a *CF* jelzőbit (carry flag), valamint az *AL* és az *AH* regiszterek kombinációjában jelenik meg. Ha nincs hiba, a *CF* = 0, ha van, akkor *CF* = 1. Az *AL* és az *AH* a hibakódot tartalmazza a két különböző és bizonyos szempontból redundáns csoportban.

A 15.2. táblázatban megadott hibakódok a DOS saját hibakódjai, amelyek a 36-os megszakítás által visszaadott kódokon alapulnak. A 15.3. táblázatban látható *AH*-beli hibakódok viszont a ROM-BIOS-tól származnak.

Általában a megszakításkezelők és egyéb szolgáltatásrutinok befejezésük után kitakarítják a vermet, visszaállítva azt eredeti méretére és tartalmára. A 37-es és a 38-as DOS-megszakítás

15.2. táblázat
A 37-es és 38-as DOS-megszakítások révén a lemezolvasási és -írási hibát követően az *AL*-regiszterben található hibakódok

Hibakód		Jelentése
Dec	Hexa	
12	0C	Általános, nem specifikus hiba
11	0B	Olvasáshiba
10	0A	Íráshiba
8	08	Nem találja a szektort
7	07	Nem ismert közeg: a formátumot nem ismeri fel
6	06	Kereséshiba: az igényelt sávra lépés meghiúsult
4	04	CRC-hiba: paritáshiba
2	02	A meghajtó nincs készenlétben (pl. nincs lemez a meghajtóban vagy az ajtó nyitva van)
1	01	Nem ismert egység: érvénytelen a meghajtószám
0	00	Írásvédelem-hiba: védett lemezre kísérelt meg írást

15.3. táblázat
A 37-es és a 38-as DOS-megszakításokon keresztül a lemezírási, ill. -olvasási hibát követően az *AH*-regiszterben található hibakódok

Hibakód		Jelentése
Dec	Hexa	
128	80	A várakozási idő letelt: a lemez nem reagált
64	40	Hibás keresés: a keresett sávra állás meghiúsult
32	20	A vezérlő hibás: a lemezegység-vezérlő nem működik
16	10	Rossz CRC: az olvasás érvénytelen, az adatrészben paritáshibát talált
8	08	A közvetlen memóriahozzáférés (DMA) hibája
4	04	Rossz szektor: a kért szektor nincs a lemezen
3	03	Írásvédelem-hiba: írásvédett lemezre megkísérelt írás
2	02	Rossz címjel: a szektorazonosító jelzés érvénytelen, ill. nem találta
0	00	Egyéb hibák

szándékosan nem ezt teszi; ehelyett egy szót hagy a veremben és így tér vissza a hívóprogramhoz. Ez a szó tárolja a jelzőbitek regiszterének tartalmát, megmutatva a jelzőbitek akkori állapotát, amikor a program meghívta a szolgáltatást. Ez szándékos, mivel a 37-es és a 38-as megszakítás a jelzőbitek regiszterén keresztül ad vissza információt, tehát elrontja az eredeti tartalmát. Valószínűleg értelmetlen ekkora elővigyázatosság, mivel bármely program, amelynek szükséges a jelzőbitek értékének megőrzése, azt teheti, amit egy program általában meg is tesz, amikor valamit elmenteni akar: **PUSH** utasítással a verembe helyezi. Bármely programnak, amelyik a 37-es és a 38-as megszakítást használja, a szolgáltatás vége után a **POP** utasítással két extra jelzőbit-státuszbájtot ki kellene vennie a veremből. Kivehetők a **POPF** utasítással is, ha a *CF* regisztert már megvizsgáltuk. Ekkor a jelzőbitek eredeti értéke visszakerül a jelzőbitregiszterbe. Ha a jelzőbiteket valamilyen külső helyre mentjük, akkor az eredeti értéket el is dobhatjuk.

A 39-es megszakítás (hexa 27): a program befejezése memóriában maradással

A 39-es megszakítás a DOS egyik legérdekesebb szolgáltatása. Annyira fontos ez a szolgáltatás, hogy a fejezet végén egy assembly nyelvű programban is felhasználtuk.

A 32-es megszakításhoz hasonlóan a 39-es megszakítás is befejezi a programot, de nem törli a memóriából, hanem a program egy meghatározott részét a memóriában hagyja (a program rezidens marad); a memória első használható címének DOS nyilvántartása pedig a rezidens programot közvetlenül követő bekezdési címre (paragraph address) változik. Az az információ, melyet a 39-es megszakítás használatával teszünk memóriarezidenssé, DOS-kiterjesztéssé válik, és más programok nem írhatják felül. Betartva az IBM következetes erőfeszítését, hogy ne használják a DOS-megszakításokat, a 49-es függvény (hexa 31) DOS-függvényhívás ugyancsak ezt a szolgáltatást nyújtja (☛ l. a 17. fejezetet).

A 39-es megszakítást (vagy annak függvényhívás-megfelelőjét) használja számos fejlett program, amely a DOS betölthető kiegészítéseként szerepel. E programok közül az egyik legismertebb a billentyűzetbővítő ProKey. A programok ezt a szolgáltatást általában arra használják, hogy új, határozatlan ideig érvényben maradó megszakításkezelő rutint hozzanak létre. Leggyakrabban ezek a megszakítási rutinok létező megszakításkezelőket pótolnak azért, hogy megváltoztassák vagy kibővítsék a műveleteiket. De a memóriarezidens-tétel nem korlátozódik a megszakításkezelőkre és a programinstrukciókra – ugyanilyen feltételek mellett adat is lehet. Ugyanezt a programozási technikát pl. használhatjuk arra is, hogy állapotinformációt töltsünk olyan közös területekre, amelyeken különböző programok osztoznak. Ezzel lehetővé tesszük számukra, hogy egymással közvetlenül kommunikáljanak.

Általában az ilyen technikát használó program csak egy részt kíván rezidensen hagyni, eldobva az inicializáló kódot. A programnak ezért már a futás megkezdésekor tartalmaznia kell a rezidens részt, és a *DX*-regiszterben meg kell határoznia a rezidens részt követő bájtnak a kódszegmens kezdetéhez viszonyított eltolási címét. (☛ L. a fejezet végén lévő programpéldát.)

E szolgáltatással memóriarezidensként hagyott bármely rész általában mindaddig rezidens marad, ameddig a DOS is rezidens. Az sem rendkívüli, hogy számos más program is rezidensként hagyja részeit. Mivel ezek a programok általában még fejlettebb, összetettebb technikát használnak, csak rendkívüli esetben fordulhat elő, hogy egymással ütköznek. A rezidensprogramok csoportjának működtetése miatt azokat néha meghatározott sorrendben kell betölteni, ezt a sorrendet viszont csak próbálgatással állapíthatjuk meg. (Valljuk be, nem tisztességes ez az eljárás a gyanútlan felhasználóval szemben!) Ha ilyen technikát használó programot írunk, nagy körültekintéssel kell eljárnunk, csak így biztosíthatjuk a program civilizált viselkedését.

A 32-es megszakítással megegyező módon a szokásos befejező szolgáltatással a DOS visszaállítja a címvektorokat a 34–36-os megszakításokra (hexa 22–24), amikor a 39-es szolgáltatást végrehajtja. Ezt a szolgáltatást nem lehet arra használni, hogy a címmegszakításokhoz rezidens megszakításkezelőket hozzunk létre. Talán ez korlátnak tűnik, valójában azonban igen ésszerű, ugyanis a címmegszakításokat nem kívánjuk globálisan használni, csak egyedi programok használatára ajánljuk (☛ l. még a DOS-címmegszakítási részben).

🕮 MEGJEGYZÉS: Amikor EXE típusú programot szerkesztünk, azt hagyományosan a legmagasabb rendelkezésre álló memóriahelyre tegyük, ne a legalacsonyabbra. Ezek a programok nem tudják használni a 39 megszakítást, mert azt csak alacsony memóriacímen futó programok számára tervezték. (☛ A szerkesztésre és az EXE típusú programokra vonatkozó további információkat l. a 19. fejezetben.)

A 47-es megszakítás (hexa 2F): a nyomtatóspooler vezérlése

🕮 MEGJEGYZÉS: E fejezet anyagának jelentős része az összes DOS-változatra vonatkozik. A 47-es megszakítás azonban csak a DOS 3.00 változatnál és a későbbi változatoknál áll rendelkezésünkre.

A DOS 3.00-val egy új megszakítást, a 47-es (hexa 2F) megszakítást vezetjük be arra, hogy a programok számára hozzáférhetővé tegyünk egy hatékony és hasznos DOS-jellemzőt, a nyomtatóspoolert.

Ez a megszakítás az a szabványos mód, amellyel bármely DOS-ban installált nyomtatóspoolerrel kommunikálhatunk; akkor is, ha az a DOS **PRINT** utasítás nyújt szabványos nyomtatóspooler, és akkor is, ha nem. A 47-es megszakításban a rendelkezésünkre álló nyom-

tatóspooler vezérlő szolgáltatásai hat különböző függvényből épülnek fel. Ezeket a 0-tól 5-ig kódolt függvényeket úgy használjuk, hogy először egy függvénykódot helyezünk az *AL* regiszterbe, és utána kiadjuk a 47-es (hexa 2F) megszakítást.

A 0-s függvény jelzi, hogy van-e nyomtatóspooler. A választ az *AL* regiszterben kapjuk vissza. A 255 (hexa FF) érték jelenti azt, hogy van nyomtatóspooler, és feltehetően használható. A 0 érték jelzi, hogy jelenleg nincs, de csatolható; az 1 érték pedig azt, hogy nincs és nem is lehet.

A részletesebb vizsgálatok előtt ez a két utolsó visszaadott kód elég különösnek tűnhet. A 0 visszaadott érték üzenete a nyomtatóspoolerről így hangzik: „Nem vagyok itt." Erre azt válaszolhatnánk: „Ha nem vagy ott, hogyan tudsz válaszolni". Úgy, hogy a 0 válasz automatikus, amikor nincs olyan megszakításkezelő, amely a kérésünkkel foglalkozna. Az 1 értékű visszaadott válasz mintha azt mondaná: „Itt vagyok, de nem használhatsz engem". Bármennyire furcsa is, tulajdonképpen erről van szó. Az 1 értékű visszaadott kód azt jelenti, hogy a nyomtatóspooler alkalmazása tilos, mivel a 47-es megszakítást valamely más megszakításkezelő valamilyen más célra használja. Elbűvölő mindezt végiggondolni.

Az 1-es funkciókóddal egy állományt nyomtatásra jelentünk a nyomtatóspoolernek. Meg kell mondanunk, hogy mi a nyomtatandó, ezért a *DS*:*DX* regiszterpárt úgy állítjuk be, hogy az a *bejelentett csomag*nak nevezett 5 bájtos területre mutasson. A bejelentett csomag első bájtja egy szintkód (amelyről semmit sem tudunk), a bejelentett csomag megmaradó négy bájtja pedig egy ASCIIZ-string szegmentált címe (☛ l. a 17. fejezetet), amely a nyomtatandó állomány útjának nevét határozza meg. Az útnév mindig csak egyetlen állományra vonatkozhat. A globális állománynévkarakterek, mint * és ?, nem megengedettek.

Ha ezt a függvényt használva egy állományt bejelentünk, akkor azt a nyomtatandó állományok sorának, ill. listájának végére írjuk. Az állományokat a függvény egymás után nyomtatja ki, és nyomtatás után törli a sorból.

A 2-es függvénykód törli a kinyomtatásra váró állományokat. A *DS*:*DX* regiszterpár arra az ASCIIZ-stringre mutat, amely meghatározza, hogy melyik állományt kell törölni a sorból. Ebben az esetben a * és ? globális állománynév-karaktereket lehet használni. Meg kell jegyeznünk, hogy a *DS*:*DX* a 2-es függvényben közvetlenül az állománynévstringre mutat, az első függvény viszont az előterjesztett csomagra mutat, és csak utána a stringre.

A 3-as függvény törli az összes nyomtatásra váró állományt. Mint a 2-es és 3-as függvénynél, ha az éppen kinyomtatás alatt levő állomány törlődik, a DOS megállítja a nyomtatást és ezzel összhangban egy rövid üzenetet küld ki.

A 4-es függvénykód lehetővé teszi a programoknak, hogy a nyomtató sort megfigyelhessék. Amikor ezt a függvényt kérjük le, a sor befagy, tehát nyugodtan szemlélhetjük azt, nem kell amiatt aggódnunk, hogy eközben a lista megváltozik. Bármely más 47-es függvényhívás kibocsátása a sort „felolvasztja". A 4-es függvény egy mutatót ad vissza a *DS*:*SI* regiszterpárban, amely a nyomtatásra sorba állított állománynevek listájára mutat. A lista tagjai 64 bájtos kötött hosszúságú stringek. A lista végét 0 bájttal kezdődő tag jelzi.

A 4-es függvény által előidézett sorbefagyasztás nem állítja le a nyomtatási műveletet, erre nincs is szükség, de felfüggeszti azoknak az állományoknak a sorból való törlését, amelyek nyomtatása befejeződött.

Az 5-ös függvénykód lényegében 0 függvény, amely nem csinál semmit, csupán a 4-es függvény által befagyasztott állománynévlistát „felolvasztja" (erre a másik négy függvény is képes).

A három DOS címmegszakítás

A DOS a 34 – 36-os (hexa 22 – 24) terjedő három megszakítást használja arra, hogy három kivételes körülményt kezeljen: a program végét, a billentyűzetről jövő megszakítást (*Ctrl-Break* vagy *Ctrl-C* a szabvány PC billentyűzeten; *Fn-B* a PCjr-en), valamint bármilyen „kritikus hibát" (általában valamilyen lemezhibát). Programunk befolyásolhatja e három körülmény mindegyikében a követett eljárást azáltal, hogy a megfelelő megszakítási vektort bármelyik általunk választott műveletre irányítja. Ezért hívjuk ezeket a megszakításokat címmegszakításoknak.

A DOS e megszakítások mindegyikére fenntart egy alap címbeállítást, amelyet a program kezdeténél megőriz és visszaér, ha a program befejeződött. Ez lehetővé teszi a programjainknak, hogy szükségleteik szerint ezeket a vektorokat szabadon változtassák anélkül, hogy megzavarnák a következő programok vagy a DOS műveleteit.

Megváltoztathatják programjaink a megőrzött alapbeállításokat is, amely ezért egy „félig állandó" változást okoz a DOS-műveletben. A program az alapbeállításokat a programszegmens előtagban tárolja (következőkben részletezzük). A PSP-ben levő érték módosítása automatikusan megváltoztatja azt az alapbeállítást, amelyet a program befejezésekor visszaállít.

Általános és elfogadott gyakorlat, hogy egy program a saját műveletei alatt megváltoztathatja a megszakításcímeket; nem szokás viszont az alapbeállítást megváltoztatni, mert az érvényben marad a program végéig.

A 34-es megszakítás (hexa 22): a befejezési cím

A 34-es megszakításhoz kapcsolódó cím azt határozza meg, hogy a számítógép vezérlése hová kerül a program lefutása után. Ezt a címet a program bemásolja a PSP-jébe is. Szokás ezt a szolgáltatást arra használni, hogy a vezérlést visszaadjuk a DOS-utasítás interpreterének, a COMMAND.COM-nak, amikor a program véget ért. Ennél a résznél a másik két megszakításvektort programunkkal szabadon változtathatjuk, hogy új rutinokra mutassanak, ezt a vektort azonban ilyen módon nem használhatjuk. Ennek a vektornak a használatát a legjobban egy példán keresztül tudjuk elmagyarázni. Egy program – hívjuk Prog1-nek – kérheti, hogy a DOS töltsön be és futtasson egy másik programot, amelyet Prog2-nek hívunk. Amikor a Prog2 véget ér, a DOS visszaadja a vezérlést bárhova, ahova a 34-es megszakítás alapbeállítása érvényes, akkor a vezérlést a DOS-nak adja vissza és nem a Prog1-nek. Lehet, hogy ez éppen az, amit a Prog1 is akar, de, ha a Prog1 a Prog2 befejezése után is folytatni akarja a működését, akkor a Prog1-nek úgy kell megváltoztatni a befejezési cím megszakítási vektorát, hogy az a Prog1-en belüli helyre mutasson, mielőtt futtatni kezdené a Prog2-t. Később a Prog1-nek ezt a megszakítási vektort vissza kell állítani, hogy a Prog1 a szokásos módon fejeződjön be, visszaadva a vezérlést a DOS-nak.

A többi címmegszakítástól eltérően ezt a megszakítást valójában sohasem generáljuk. Ehelyett a megszakításvektort a szegmenscím tárolására használt helyként kezeljük. A 29 – 31-es, a 68-as és a 73-as (hexa 1D – 1F, 44 és 49) megszakításokat hasonlóan címek tárolására használjuk (☛ l. a 3. fejezetben). Igazán különleges ez az eljárás, ezért ha nem értjük tökéletesen, inkább ne bonyolódjunk bele. Ha elég képzettek vagyunk, és ezeket a jellemzőket tudjuk használni, akkor valószínűleg a magyarázatoknál jobban megértjük majd.

A 35-ös megszakítás (hexa 23): a megszakításkezelő címe

A 35-ös megszakítással társult cím a megszakításkezelő rutinra mutat, mely minden olyan esetben meghívásra kerül, amikor a DOS egy billentyűzetről jövő megszakítási műveletre válaszol. A megszakítást a szabvány PC-billentyűzeten a *Ctrl-Break*-kel, bármely egyéb billentyűzeten pedig a *Ctrl-C*-vel hozzuk létre. (Ezt a tényt nem hirdetik széles körben!) A DOS kicsit csalafintán válaszol egy-egy billentyűzetről jövő megszakítási műveletre. Szabványműveletben a DOS csak bizonyos billentyűzet-, ill. képernyőfunkcióknál válaszol egy megszakításra. A **BREAK ON** utasítás a DOS 2.00 és a magasabb változatok számára mégis lehetővé teszi, hogy bármikor válaszoljon egy megszakításra.

A DOS alapválasza a megszakításra az, hogy beszünteti a program vagy a kötegelt utasításállomány futtatását. Ha a programjaink beállították saját megszakítási rutinjukat, bármilyen általuk kívánt akciót elvégeztethetnek a DOS-szal, attól függetlenül, hogy az mennyire kiterjedt vagy milyen összetett. Ezzel a megszakítással bármilyen DOS-szolgáltatást elérhetnek programjaink, nem szükséges a vezérlést visszaadniuk a DOS-nak. (Pedig ezt kellene tenniük a veremnövekedés elkerülése céljából.) A leggyakoribb két tevékenység: a megszakítási műveletet teljesen figyelmen kívül hagyni, vagy egy ismétlődő műveletből való kilépéshez jelként használni. Számos DOS-program szemlélteti ezt a második felhasználást. A kezdetleges Edlin szövegszerkesztő program pl. ezt a megszakítást annak jelzésére használja, hogy befejezi az I-utasítást (sorbeszúrás). Ha azt akarjuk, hogy a program figyelmen kívül hagyja a megszakítóbillentyűzetet, használjunk olyan megszakításkezelőt, amely a vezérlést az aktív programnak adja vissza. Ha a megszakítási billentyűt egy ismételt művelet befejezésének jelzésére használjuk, a megszakításkezelő kijelöl egy adatjel-kapcsolót, amelyre a program másik része figyel, és amelyen valamilyen tevékenységet végez, majd visszaadja a vezérlést.

Két módon adhatja vissza a megszakításkezelő a vezérlést. Szabályos eljárás bármely megszakításkezelő esetében az **IRET** (interrupt return) utasítás használata. A másik sajátos eljárást az jellemzi, hogy a megszakításkezelőnek választási lehetőséget ad, közölve a DOS-szal: vagy folytassa, vagy fejezze be a programot. (Ugyanúgy, ahogy az alap DOS-megszakításkezelő leállítja a programokat, a megszakítási billentyűzetműveletekre való válaszként.) A módszer használatakor a megszakításkezelő beállítja a *CF*-et (carry flag), és a **FAR RET** utasítással be is fejezi; ha $CF=1$, akkor jelzi, hogy a DOS-nak ki kell lépnie a programból, ha $CF=0$, akkor a programot folytatni kell.

🕮 MEGJEGYZÉS: A programok ezen megszakítás meghívása révén szimulálhatják a billentyűzet megszakítási műveletét.

A 36-os megszakítás (hexa 24): a kritikus hibakezelő címe

A 36-os megszakításhoz kapcsolódó cím arra a megszakításkezelő rutinra mutat, amely akkor lép működésbe, ha a DOS egy „kritikus hibát", vészhelyzetet fedez fel, amely megakadályozza a program folytatását. A kritikus hiba általában lemezhiba, de más hibát is jelenthet, mint ezt később látni fogjuk.

Ha egy hibakezelő meghívására kerül sor, a hibáról magáról és a hiba előfordulása előtti állapotról számos információ áll rendelkezésre. Ez tartalmazza a *BP : SI* regiszterpárt, a vermet, az *AH* regisztert és a *DI* regisztert. A folyamat összetettsége miatt egyenként fogunk velük foglalkozni.

Ha a DOS 2.00 vagy egy magasabb szintű változata alatt dolgozunk, a *BP : SI* a készülékfejléc-vezérlőtömbre mutat. A hibakezelő ezt a vezérlőtömböt meg tudja figyelni azért, hogy

többet tudjon meg a hibás készülékről (lemezegységről, nyomtatóról stb.). A további információkat l. a DOS Technical Reference kézikönyv készülékfejlécre vonatkozó részében.

A verem tartalmazza annak a programnak a teljes regiszterkészletét, amely a kritikus hibával végződött DOS-funkcióhívást kiadta. Ez az információ az aktív programba közvetlenül beépülő hibakezelőnek igen hasznos lehet. Feltételezve, hogy a hibakezelő hagyományos módon fog hozzáférni a veremhez, egy relatív címen meghatározhatjuk a verem helyét a *BP*-ből a következő két utasítással:

```
PUSH BP
MOV BP,SP
```

Ezek közül az első elmenti a régi *BP*-t (base pointer), aztán a *BP*-t az *SP*-vel (stack pointer) teszi lehetővé. (☛ Ezekre az utasításokra a további információkat l. a 8. fejezetben a beviteli kódok című résznél.) Ezt elvégezve, a 15.4. táblázatban bemutatott veremtartalmakat találjuk.

A kritikus hiba jellegét az *AH* regiszter magas helyiértékű bitje és a *DI* regiszter alacsony helyiértékű bájtja jelzi elsődlegesen (egy biztos, furcsa egy választás). Ha az *AH* magas helyiértékű bitje 0 ($AH < 128$), akkor a hiba lemezműveletre vonatkozik; de ha ugyanez a bit 1 ($AH > 127$), akkor a hiba valami más lehet, amint ezt rövidesen látni fogjuk. Ha a hiba egy lemezmeghajtó hibája ($AH < 128$), az *AL* regiszter megadja a meghajtó azonosítóját (0 az *A* meghajtó, 1 a *B* meghajtó stb.). Az *AH* 0-tól 2-ig terjedő bitjei további információt adnak a hibáról, ezt a 15.5. táblázat mutatja.

Ha az *AH* nagyobb, mint 127, akkor a hiba nem szükségszerűen lemezhiba, bár lehet az is. Egy olyan lemezhiba, amelynek észlelésekor *AH* értéke nagyobb mint 127, a lemez FAT-jában lévő hiba. A DOS 1 változatoknál ez mindig igaz. A DOS 2.00-nál és az újabb változatoknál a hibakezelő programnak felügyelnie kell a készülékfejléc-vezérlőtömböt (melyre a *BP : SI*-vel mutatunk). Ha a visszaadott érték azt mutatja, hogy a készülék egy lemezegység, akkor a hiba FAT hiba. Ettől az egy kivételtől eltekintve az $AH > 127$ a lemezegységtől független hibát

15.4. táblázat
Kritikus hibában végződött DOS-funkciót meghívó programok veremtartalma

BP-hez viszonyított eltolási cím	Veremtartalmak
0	Az a *BP*-érték, amelyet elmentettünk
2	Az e kezelőt kiváltó DOS-szolgáltatás *IP : CS*-e
6	Az e kezelőt meghívó DOS-szolgáltatás jelzőbitjei
8	A DOS-szolgáltatást meghívó *AX* regisztere
10	A DOS-szolgáltatást meghívó *BX* regisztere
12	A DOS-szolgáltatást meghívó *CX* regisztere
14	A DOS-szolgáltatást meghívó *DX* regisztere
16	A DOS-szolgáltatást meghívó *SI* regisztere
18	A DOS-szolgáltatást meghívó *DI* regisztere
20	A DOS-szolgáltatást meghívó *BP* regisztere
22	A DOS-szolgáltatást meghívó *DS* regisztere
24	A DOS-szolgáltatást meghívó *ES* regisztere
26	A DOS-szolgáltatást meghívó *IP : CS* regiszterpárja
30	A DOS-szolgáltatást meghívó jelzőbitjei

15.5. táblázat

A 36-os megszakítás után az *AH*-regiszter alsó három bitjében küldött bitértékek és az azokhoz kapcsolódó hibák

Bit 2	Bit 1	Bit 0	Érték	Jelentés
.	.	0	0	Olvasáshiba
.	.	1	1	Íráshiba
0	0	.	0	A hiba DOS rendszerfájlt érint
0	1	.	1	A hiba a FAT-ot érinti
1	0	.	2	A hiba a tartalommutató hibája
1	1	.	3	A hiba a lemez adatterületét érinti

jelez. A tényleges probléma meghatározása szempontjából az alap hibakezelő kódra kell támaszkodnunk, amelyet a *DI* alacsony helyiértékű bájtján találhatunk (a magas helyiértékű bájtot figyelmen kívül kell hagyni).

A 15.6. táblázaton bemutatott *DI* hibakódértékek lényegében ugyanazok, amelyeket a 37-es és a 38-as megszakításokkal az *AL*-ben találunk.

A körülményektől függően egy hibakezelő igényelhet néhány DOS függvényhívás szolgáltatást, hogy jelentse a program használójának a történteket. A 0-tól 12-ig (hexa 0 – C) számozott függvényekkel az egyszerűbb billentyűzet- és képernyő-szolgáltatásokat lehet használni, de azokat a magasabb kódszámú szolgáltatásokat, amelyekhez legtöbb esetben lemez- vagy egyéb perifériaműveletek tartoznak, nem használhatjuk. Zűrzavart okozhat a magasabb sor-

15.6. táblázat

A 36-os megszakítás után a *DI*-regiszterben lévő hibakódértékek

Hibakód Dec	Hibakód Hexa	
12	0C	Általános, nem specifikus hiba
11	0B	Olvasáshiba
10	0A	Íráshiba
9	09	A nyomtató papírhiány hibája
8	08	Nem találja a szektort
7	07	Nem ismert a közeg: a lemezformátum nem ismerhető fel
6	06	Kereséshiba: a kért sávra állás meghiúsult
5	05	Hibás a kérési szerkezet hossza
4	04	CRC-hiba
3	03	Ismeretlen utasítást kért
2	02	A meghajtó nincs kész (pl. nincs lemez vagy az ajtó nyitva)
1	01	Nem ismert egység: érvénytelen lemezmeghajtószám
0	00	Írásvédelmi hiba: védett lemezre megkísérelt írás

15.7. táblázat
Az *AL*-regiszterbe beírható DOS-t vezérlő kódok

AL	DOS-tevékenység
0	Hagyja figyelmen kívül a hibát, és folytassa a munkát!
1	Próbálja meg újra a műveletet! (Talán megjavítottuk a hibát.)
2	Törölje a programot! (A DOS a 35-ös megszakítást (hexa 23) adja ki, miközben egy billentyűzet megszakítási műveletet hív meg.)

számú szolgáltatások használata, ha az előző műveletre vonatkozó hibakezelő közepén vagyunk. Nem valószínű, hogy a DOS talpra tud állni.

Szokásos esetben a hibakereső rutin visszatér a DOS-hoz, miután elvégezte azt, amit elvégezni jónak látott. Ezután a DOS három dolgot tehet: figyelmen kívül hagyhatja a hibát, megpróbálhatja a műveletet ismét, vagy leállíthatja a programot. A 15.7. táblázaton bemutatott értékek egyikének *AL* regiszterbe töltésével közöljük a DOS-szal, hogy mit válasszon.

🕮 MEGJEGYZÉS: Mivel az e megszakítás meghívása előtti eljárás igen összetett, nem helyes, ha a programunk a 36-os megszakítás aktiválása révén szimulál kritikus hibát.

A programszegmens előtag (PSP)

Amikor a DOS egy programot betölt, először egy memóriarészt tesz félre a programnak, amelyet programszegmensnek vagy kódszegmensnek hívunk. Ezután az első 256 (hexa 100) bájtban egy ún. programszegmens előtagot vagy PSP-t hoz létre. A program rendszerint közvetlenül a PSP után hexa 100 eltolással töltődik. A PSP információkoktélt tartalmaz, amelyet a DOS a programfuttatás támogatására használ. Ez minden DOS-programnak része, tekintet nélkül arra, hogy a program milyen nyelvben íródott. Ennek ellenére a programozási célok szempontjából a PSP-ben tárolt információk inkább az assembly nyelven írt programokra, mint a magasabb szintű programnyelveken készültekre vonatkoznak. Ez azért van így, mert a magasabb szintű nyelvekben a nyelv rendszerint felelős a program működési környezetéért, a memóriahasználatért, az állományvezérlésért; minden olyan dologért, ami a PSP-t érintheti. Rendszerint akkor tudjuk jól felhasználni a PSP-t, ha a programunk assembly nyelvi alapú – bár amelyik program ki tudja találni a *CS*-regiszter beállítását, használhatja ezt a PSP-hez való hozzáférésre (☛ l. a fejezet végén lévő programpéldát.)

Mielőtt leírnánk a PSP különböző elemeit, szemügyre kell vennünk a PSP és az általa támogatott program közötti viszonyt.

A PSP a kódszegmensen belül mindig 0 eltolással helyezkedik el. Bár a program általában közvetlenül a PSP után a hexa 100 eltolással kezdődik (ez változhat a használt programutasítástól függően). Tekintet nélkül a helyére, amint a program vezérlést kap, bizonyos regiszterek beállítódnak, hogy a PSP-re mutassanak. Az egyszerű .COM formátum programoknál az

összes szegmens regiszter úgy állítódik be, hogy a PSP elejére mutasson, és a program hexa 100 eltolással kezdődik. Egy összetettebb .EXE formátumú programnál, amely a DOS **LINK** műveletet használja, csak a *DS* és az *ES* regiszterek mutatnak a PSP-re. A **LINK** program megkerüli a *CS*, az *IP*, az *SS* és a *SP* regisztereket, emiatt a program kezdetét a *CS* : *IP*-ben a hexa 100 eltolástól eltérő helyre is lehet tenni.

Bárhová is van a program helyezve, a lényegi kapcsolat a program és annak PSP-je között ugyanaz marad. Fontos, hogy a programvégrehajtás kezdetén a szegmensregiszterek egyikével hozzáférjünk a PSP-hez. Ez azt jelenti, hogy ha a programvégrehajtás folyamán a regiszterek változnak is, a program korai fázisában kaphatunk egy mutatót a PSP-re, amely a program végrehajtás alatt biztosítja a PSP-hez való hozzáférés fenntartását.

A legjobban akkor tudjuk elmagyarázni a PSP és a program együttműködését, ha közvetlenül rátérünk a PSP belső szerkezetére. A következőkben minden elem célját és lehetséges felhasználását ismertetjük.

A PSP belső struktúrája

Hamarosan látni fogjuk, hogy a PSP különböző mezők zavaros egyvelegét tartalmazza (l. 15.8. táblázat). A DOS háttere és története két különböző irányba húzza azt – egyik vissza a korábbi CP/M rendszerekhez, a másik pedig előre, a UNIX típusú operációs rendszerkörnyezetekhez. Ennek eredményeként a PSP olyan elemeket tartalmaz, amelyek különböző célokat

15.8. táblázat
A programszegmens előtag (PSP) részei

Eltolás				Leírás
Mező	Dec	Hexa	Hossz	
1	0	0	2	Az **INT 32**-utasítás
2	2	2	2	A memória mérete (bekezdésekben)
3	4	4	1	Fenntartva; általában 0
4	5	5	5	Hívás a DOS-funkció feladóhoz
5	10	A	4	Befejezési vektor
6	14	E	4	Megszakítási vektor
7	18	12	4	Hibavektor
8	22	16	22	A DOS használja
9	44	2C	2	A környezetet leíró string
10	46	2E	34	DOS-munkaterület
11	80	50	3	**INT 33**, **RETF**-utasítások
12	83	53	2	Fenntartva
13	85	55	7	FCB#1 bővítés
14	92	5C	9	FCB#1
15	101	65	7	FCB#2 bővítés
16	108	6C	20	FCB#2
17	128	80	1	Paraméterhossz
18	129	81	127	Paraméterek
19	128	80	128	A lemezes adatátvitel területe (DTA)

szolgálnak, és különböző programozási módszerekre orientáltak. Ne jöjjünk ettől zavarba, az ilyen előtörténetű „öszvér" dolgoknak, mint a PSP is, egyszerűen ilyen a természete.

Az egyes elemeket abban a sorrendben tárgyaljuk, ahogy megjelennek.

Az *1. mező* tartalmazza a hexa CD és a hexa 20 bájtokat, a 32-es (hexa 20) megszakítási utasítást. Ebben a fejezetben a 32-es megszakítás tárgyalásakor már láttuk, ez az utasítás csak egyike azoknak a szabványos módoknak, amellyel egy program a futását be tudja fejezni. Ez az utasítás a PSP elején (a 0 eltolásnál) helyezkedik el; a program véget tud érni egyszerűen úgy, hogy erre a helyre ugrik, amikor a *CS* a PSP-re mutat. Természetesen nem ez a legésszerűbb egy program számára. Célszerűbb, ha végigmegyünk a megfelelő megszakításon vagy függvényhíváson. Feltételezhetjük, hogy ezt a furcsának tűnő rendszert biztonsági meggondolásból tervezték. Ha egy kapcsolt programnak egy meg nem oldott külső referenciája van, egy 0 eltolású hívásba torkollik, azaz erre az utasításra lép, és befejezi a programot.

A 2. mező közli a DOS memóriaterület végét jelentő bekezdés szegmenscímét. Ebből látható, hogy mennyi memóriát használhatunk. Ezt a számot 16-tal kell megszoroznunk, hogy megadja az összes bájt számát, amelyet a DOS felhasználhat. Ugyanezt az összeget jelzi ki a **CHKDSK** DOS-utasítás. Ebben a mezőben különösen a jelzett memóriaösszegre figyeljünk, mert lehetséges, hogy az nem a memória tényleges fizikai mérete; pl. számos RAM-diszk a felső memóriahelyeket használja, átállítva a DOS azon feljegyzését, amely a használható memória végét adja meg. Minden programnak, amelynek a rendelkezésre álló teljes memóriát fel kell használni, a hasonló BIOS 18-as megszakítási szolgáltatás helyett ezzel a jelzővel tudhatja meg, mennyi a szabad terület.

A konvencionális DOS működési környezet az összes rendelkezésre álló memóriát az egyes külön futó programokhoz rendeli. Olyan ablak vagy többmaszkos környezetben, ahol az egyedi programoknak a memórián osztoszkodniuk kell más programokkal, egy-egy program annyit foglal le, amennyire szüksége van, a megmaradó részt a 74-es (hexa 4A) DOS függvénnyel adja vissza. Ez a funkció eleget tesz a memóriahasználatra vonatkozó DOS-konvencióknak, de néhány ablaktechnikájú rendszerrel nem működik tökéletesen. Ezek a rendszerek a 4. mezőt használhatják a rendelkezésünkre álló memória másik jelzőjeként.

A 4. mező több annál, mint aminek először látszik. Felületesen megítélve, egy hosszú hívás a DOS függvénykezelőjéhez, amely használható ugyan a DOS-függvények meghívására – de valójában nem szabad használni. (A DOS függvénykezelője a 33-as (hexa 21) megszakítás programjának az eleje: a hívás paraméterei alapján eldönti, hogy melyik feladatot kell végrehajtani. A lektor.) Valójában ez a mező meghatározza, hogy a programjaink számára vajon 64K-nál kevesebb memória áll-e rendelkezésre. Hívásként ez az utasítás a DOS függvénykezelő rutinjának címét szegmentált formátumban tartalmazza. Egy olyan eljárással, amely túlságosan szokatlan és összetett ahhoz, hogy elmagyarázzuk; ugyanis a szegmentált címet úgy állítják be, hogy két célt szolgáljon. Nemcsak a DOS függvénykezelőjére mutat, hanem az eltolási rész azt is mutatja, hogy a kódszegmensből mennyit használhatunk (a hexa FFF0 16 bájttal kevesebb, mint 64K). A cím eltolási része a minket érdeklő rész a PSP-n belül, az 5. eltolásnál levő utasítás kódja mögött a 6. eltolásnál van.

Ennek az a következménye, hogy ha a DOS-nak 64K-nál kevesebb áll rendekezésére, amit a programunknak átadhat, akkor ebből a mezőből megtudhatjuk a rendelkezésünkre álló bájtok számát. Ennek a technikának az ablak, ill. többtaszkos rendszerek legtöbbjével vagy mindegyikével működnie kell. Ha a DOS több mint 64K-t tud adni, akkor a 2. mező, azaz a paragrafuscím felső korlátjának megfigyelésével tudjuk meg, hogy mennyivel többet. Már korábban is említettük, hogy ez a mező néhány ablaktechnikás vagy többtaszkos környezetben nem a pontos választ adja.

Az 5., a 6. és a 7. mező a már tárgyalt három cím (programbefejezés, programmegszakítás, hibakezelés) alapértékét tartalmazza szegmentált formában. Az alapcímeket a programművelet

elején megőrizzük, a program végén pedig visszaállítjuk. Talán emlékszünk még, ez lehetővé teszi számunkra a program futása alatt az eltérő szolgáltatásvektorok használatát anélkül, hogy a következő programokat vagy a DOS működését zavarnánk. Ha nem adunk új vektorokat, a DOS a PSP-ben tárolt alapbeállításokat használja, hogy az alapszolgáltatás rutinokra mutasson. Ha e mezőben tárolt címekkel manipulálunk, megváltoztathatjuk az alapértékeket (és a hozzájuk kapcsolódó szolgáltatásrutinokat), amelyek a program befejezésekor visszaállításra kerülnek. Ezáltal egy fél permanens változást hozunk létre a DOS műveletben. Szokásos körülmények között nincs elfogadható ok arra, hogy megváltoztassuk vagy kiolvassuk ezeket a beállításokat.

A 9. mező azt a szegmenscímet tartalmazza, amely a DOS 2.00, ill. a későbbi változatok által támogatott környezetstringekre mutat. A környezet ettől a címtől számított 0 eltolással kezdődik.

🕮 MEGJEGYZÉS: A „környezet" szó ebben az összefüggésben a stringkészletre utal (amelyet azonnal meghatározunk) azért, hogy a terminológiában elkerüljük a zavart. A „környezet" kifejezést a DOS arra használja, hogy a programokkal bizonyos információkat közöljön. Máshol a „környezet" szót arra használtuk, hogy utaljunk azokra a konvenciókra, amely alatt egy program működik.

A DOS-környezet ASCIIZ-stringek gyűjteménye, azaz olyan ASCII karaktereké, amelyek **CHR$(0)** jellel lezártak és különböző információkat határozhatnak meg. Minden egyes környezetbeállítás végét egy zéróbájttal (**CHR$(0)**) jelezzük, azt várva, hogy így megtaláljuk a következő string első bájtját. Ha egy környezetbeállítás **CHR$(0)**-val kezdődik, akkor nincs benne string.

Konvenció szerint minden egyes környezeti string NAME = érték formátumban van, ahol a NAME csupa nagybetű, és bármilyen ésszerű hosszúságú, az érték pedig csaknem bármi lehet. A DOS beállítja az utasításprocesszor számára, majd átadja az összes futtatandó programnak. Szokásos esetben a környezet a COMSPEC nevet használja (ezzel találja meg a DOS a lemezen a COMMAND.COM fájlt), valamint tartalmazhat még olyan neveket is, mint a PATH vagy SWITCHAR. A **SET** DOS-utasítást a stringek környezethez való hozzáadására, módosítására vagy törlésére használhatjuk.

A 11. mező két utasítást tartalmaz, amely egy DOS-függvényt (33-as megszakítás, hexa 21) hív meg, és visszatér a hívóhoz (**RETF** vagy **FAR** visszatérés). Ez újabb trükk, amely lehetővé teszi, hogy „fél-közvetve" DOS-függvényeket hívjunk meg. Ahhoz, hogy ezt a lehetőséget használjuk, a DOS-függvény meghívásához szükséges minden feltételt be kell állítani (a függvényt az *AH*-ban kiválasztjuk stb.). Ezután ahelyett, hogy csak egyszerűen a 33-as megszakítást hajtanánk végre (egy 1-bájtos utasítás), egy távoli hívást adunk ki a PSP-ben a hexa 50 eltolásra (egy 5 bájtos utasítás).

Azt gondolhatnánk, hogy ez a lehetőség egy újabb felvillanás a múltból, egy darabka CP/M kompatibilitás; de ezt valójában a DOS 2.00-val vezették be, az előző DOS változatokban nem is működik. Feltételezhetjük, hogy a DOS szolgáltatások meghívásának ez a megközelítése valóban olyan esetlen, mint amilyennek tűnik, mégis a jövőbe tekint, és nem a múltba.

A 13., a 14., a 15. és a 16. mezők az állományvezérlő tömböket vagy az FCB-ket használó régi típusú állományfeldolgozást támogatják. Az FCB-ket használhatjuk bármely DOS-változatban állomány I/O-ra, de szeretnénk mindenkit lebeszélni arról, hogy a 2.00 vagy későbbi változatnál ezt használja, hiszen ott már sokkal jobb állománykezelési lehetőség is van. (☛ L. az állományvezérlő tömbökre vonatkozó további információinkat a 16. és a 18. fejezetben.)

A PSP-nek ezt a területét azért hozták létre, hogy egyszerűbbé tegyék azon programok életét, amelyek egy vagy két fájlnevet kapnak paraméterként. Leleményes ötletként lehetővé

tették, hogy a DOS a szükséges FCB-ket az első két programparaméterből alkossa (a paraméterek a parancssorban a programnevet követően találhatók). Ha a programnak szüksége van az egyik vagy mindkét FCB-re, megnyithatja és használhatja azokat a programparaméterek dekódolása és az FCB megalkotása nélkül.

Ha a PSP-nek ezt a jellemzőjét használjuk, néhány problémával szembe kell néznünk. Először is, a két FCB átlapolja egymást. (Ez CP/M örökség. A lektor.) Egyszerű, ha a programunknak csak az elsőre van szüksége, de ha a másodikra is, akkor az egyiket vagy mind a kettőt máshová kell helyezni, mielőtt használnánk. Tudnunk kell azt is, hogy ezek az FCB-k kiterjesztéseket vonhatnak maguk után, ezt a tényt viszont a PSP-re vonatkozó dokumentációk legtöbbje figyelmen kívül hagyja. A 15.8. táblázaton az FCB kiterjesztésekből adódó helyek dokumentálásával felhívjuk a figyelmet erre.

Ne felejtsük el, hogy az FCB-k használatát sokan már kissé elavultnak tekintik, de ha mégis használni akarjuk azokat, ez az információ segítségünkre lehet.

A 17. és a 18. mező hozzáférhetővé teszi programjaink számára az utasítássoron bevitt paramétereket. A 17. mező megadja a paraméterstring teljes hosszát (ez lehet 0 rövidségű vagy 127 hosszúságú). A 18. mező a tartalmakat adja meg.

Íme az átadott string néhány sajátossága: nem tartalmazza a meghívott program nevét; ehelyett egy karakterrel kezdődik, amely közvetlenül a program nevét követi, és rendszerint üres. A szeparátorokat – a helyközöket és a vesszőket – nem szűri ki és nem tömöríti. Ha az utasítássort használjuk, fel kell készülnünk arra, hogy végigtapogassuk azt, felismerve a szabványszeparátorokat. A DOS a 2.00 változatával kezdődően különösen kezeli az utasítássort: minden átirányító paramétert (<INPUT vagy >OUTPUT) kivon, és egy olyan paramétersort készít, amelyben ezek a paraméterek nem szerepelnek. Ennek az utasításstringre vonatkozó két műveletnek a következményeként a program nem tudhatja meg, hogy a szabvány I/O át lett irányítva, sőt még a saját nevét sem.

MEGJEGYZÉS: Mivel a 17. és a 18. mező átlapolódik a 19. mezővel, a PSP-ben minél előbb kérdezzük le a paramétereket, mert a következő mező kitörölheti azokat.

A 19. mező a DOS lemezátviteli területének (DTA) alapértéke. Ez egy 128 bájtos terület, amely a PSP-ben hexa 80 eltolással kezdődik. Azokhoz a DOS-szolgáltatásokhoz használhatjuk, amelyek a DTA-hoz küldenek hívásokat, amikor mi még nem állítottuk fel saját pufferterületünket. (☛ Azokat a szolgáltatásokat, amelyek a DTA-t használják vagy manipulálják, a 16. és a 17. fejezetben láthatjuk.)

A PSP-re mint egészre tekintve megjegyezhetjük, hogy a DOS számos területet használ a PSP 256 bájtjának első 92 bájtjából (1–13. mező, hexa 0–58 eltolásig). Ha a PSP bármelyik mezőjét akarjuk módosítani, korlátoznunk kell magunkat a 14–19. mezőkre (hexa 5C–FF eltolás). A következő program illusztrálja, hogy hagyhatjuk a PSP első részét érintetlenül, mialatt a második részt módosítjuk, és azt a rezidens program használja.

Programpéldák

E fejezet interfész példájához olyan programot alkotunk, amely a 39-es (a program befejezése memóriában maradással) DOS-megszakítást használja. A legtöbb assembly nyelvű példánk kapcsolatot teremt egy magasszintű nyelv és a ROM-BIOS vagy a DOS szolgáltatásrutinjai között. Példánk ettől teljesen eltér. Ez egy önálló assembly nyelvű program, amely a 39-es megszakítást használja, hogy egy részét memóriarezidenssé tegye.

Mivel néhány olyan dologgal is találkozunk, amelyről eddig még nem szóltunk, helyénvaló néhány megjegyzés. Az előző részből már tudjuk, hogy ha futtatáshoz egy program betöltődik, a DOS annak elején PSP-t hoz létre. A PSP hossza 256 (hexa 100) bájt, és a betöltött program rögtön ezután következik. A *CS* regiszter a PSP-re mutat, ezzel a program számára megkönnyíti a hozzáférést a PSP-ben levő információkhoz. Minden program tudja, hogy az indulási pontjához képest az *IP* eltolása mindig hexa 100. Ezt a tudást akkor hasznosítjuk, amikor a *CS*-t arra használjuk, hogy kiszámolja a programunkban levő memóriarezidens rész eltolását a következő sorral:

```
RESIDENT_OFFSET EQU START_RESIDENT - BEGINNING + 100H
```

Általánosságban a 256 bájt elfecsérlése, ha egy rezidens program a PSP-t a helyén hagyja, pedig a DOS-nak szüksége van a PSP-re a befejezési eljárás végrehajtásához. Két dolgot tehetünk, hogy elkerüljük ennek a területnek az elvesztését. Először: ha a rezidens programnak megszakításkezelőként az újraaktiválásakor adatterületre van szüksége, használhatja a PSP területet, mert a DOS-nak már nem kell, hiszen a programbefejezés műveletét már végrehajtotta. Másodszor: a program rezidens részét átteheti a PSP egy részére, amelyre a DOS-nak nincs szüksége a befejezéskor. Példánkban azt mutatjuk be, hogyan lehet ezt elérni, felülírva – az első 92 (hexa 5C) bájt kivételével – az egész PSP-t. Ennek elvégzése teljesen biztonságos, de ennél több felülírás már ütközést okozhat a DOS-szal.

Ha ezzel a művelettel kísérletezünk, és a DOS **CHKDSK** funkciójával azt akarjuk megvizsgálni, mennyi memóriát használtunk fel, észrevesszük, hogy a DOS minimum további 16 bájtot fal fel a saját adminisztrációjára; nagyon kis igényekre viszont még néhány százat.

A következő példa megállapításai néhány további érdeklődésre számot tartó dolgot világítanak meg.

```
; a 39-es megszakítás, memóriában maradva fejezd be
PROGRAM   SEGMENT PARA PUBLIC 'CODE'
          ASSUME CS:PROGRAM
TBSR      PROC
BEGINNING:
; induláskor az iniciáló kódra lépünk
          JMP          INITIALIZE
          START_RESIDENT:

; itt kezdődik az, amit rezidensként akarunk hagyni a memóriában

          END_RESIDENT:
          RESIDENT_LENGTH EQU END_RESIDENT - START_RESIDENT
          RESIDENT_OFFSET EQU START_RESIDENT - BEGINNING + 100H
          PSP_AMOUNT      EQU 92        ; A PSP rész, amelyet megtartunk

; az inicializáló kód következik, eldob amikor befejezzük, de memóriarezidensként
maradunk

          INITIALIZE:

; ide kerül bármely inicializáló kód, amire szükségünk lehet
; ez a kód helyezi a rezidens részt lejjebb a PSP nagyobb része fölé:

          PUSH       CS                 ; a célszegmens beállítása...
POP       ES                            ; a szegmensregiszterben
PUSH      CS                            ; a forrásszegmens beállítása
```

```
POP      DS                            ; a megmensregiszterben
MOV      DI,PSP AMOUNT                 ; a cél eltolási rész beállítása
MOV      SI,RESIDENT_OFFSET            ; a forrás eltolási rész beállítása
MOV      CX,RESIDENT_LENGTH            ; a mozgatandó bájtok számának beállítása
CLD                                    ; az előreirány beállítása
         REP MOVSB                     ; a rezidens bájtokat a PSP-be helyezi

    ; most készen állunk a befejezésre
    ; állítsa be az első rendelkezésre álló helyet a rezidens kódot követő bájt címére

                  MOV DX,PSP_AMOUNT + RESIDENT_LENGTH
         INT 27H
TBSR     ENDP
PROGRAM  ENDS
```

16 Általános DOS-függvények

Bevezetés • 243

Az általános függvények összefoglalása • 243

A 0-s függvény: a program befejezése • 245
Az 1-es függvény: billentyűzetbemenet (echóval) • 245
A 2-es függvény: monitorkimenet • 246
A 3-as függvény: soros bemenet • 246
A 4-es függvény: soros kimenet • 246
Az 5-ös függvény: nyomtatókimenet • 246
A 6-os függvény: közvetlen billentyűzet (monitor I/O) • 246
A 7-es függvény: közvetlen billentyűzetbemenet (echo nélkül) • 247
A 8-as függvény: billentyűzetbemenet (echo nélkül) • 247
A 9-es függvény: string kiírása • 248
A 10-es függvény (hexa A): tárolt billentyűzetbemenet • 248
A 11-es függvény (hexa B): a billentyűzetbemenet állapotának ellenőrzése • 249
A 12-es függvény (hexa C): töröld a billentyűzetet és hajtsd végre! • 249
A 13-as függvény (hexa D): a lemezegység alapállapotba állítása • 249
A 14-es függvény (hexa E): az aktuális lemezegység kiválasztása • 249
A 15-ös függvény (hexa F): állomány megnyitása • 250
A 16-os függvény (hexa 10): állomány lezárása • 251
A 17-es függvény (hexa 11): az első illeszkedő állomány keresése • 251
A 18-as függvény (hexa 12): a következő illeszkedő állomány keresése • 252
A 19-es függvény (hexa 13): állomány törlése • 252
A 20-as függvény (hexa 14): soros állomány rekordjának olvasása • 252
A 21-es függvény (hexa 15): soros állomány rekordjának írása • 253
A 22-es függvény (hexa 16): állomány létrehozása • 253
A 23-as függvény (hexa 17): állomány átnevezése • 253
A 24-es függvény (hexa 18): a DOS-on belül használt függvény • 254
A 25-ös függvény (hexa 19): az aktuális lemezegység visszajelzése • 254
A 26-os függvény (hexa 1A): a lemezegység transzferterületének beállítása • 254

A 27-es függvény (hexa 1B): FAT-információ lekérdezése az aktuális lemezegységről • 254
A 28-as függvény (hexa 1C): FAT-információ lekérdezése bármely lemezegységről • 255
A 33-as függvény (hexa 21): random állomány rekordjának olvasása • 255
A 34-es függvény (hexa 22): random állomány rekordjának írása • 255
A 35-ös függvény (hexa 23): az állományméret lekérdezése • 256
A 36-os függvény (hexa 24): random rekordmezők beállítása • 256
A 37-es függvény (hexa 25): a megszakításvektor beállítása • 256
A 38-as függvény (hexa 26): programszegmens létrehozása • 257
A 39-es függvény (hexa 27): random állomány rekordjainak olvasása • 257
A 40-es függvény (hexa 28): random állomány rekordjainak írása • 257
A 41-es függvény (hexa 29): állománynév elemzése • 258
A 42-es függvény (hexa 2A): dátumlekérdezés • 258
A 43-as függvény (hexa 2B): dátumbeállítás • 259
A 44-es függvény (hexa 2C): időlekérdezés • 259
A 45-ös függvény (hexa 2D): időbeállítás • 259
A 46-os függvény (hexa 2E): a lemezre írás ellenőrzésének beállítása • 259

Az állományvezérlő tömb (File Control Block, FCB) • 260

Egy példaprogram • 263

Bevezetés

Ebben a fejezetben azokat a függvényeket tárgyaljuk, amelyek általános érvényűek az összes DOS-változatra. A DOS terminológiában ezeket a régi (általános) szolgáltatásokat „hagyományos függvényeknek" nevezték. A DOS 2.00-val bevezetett új szolgáltatásokat, amelyeket a 17. fejezetben tárgyalunk, kiegészített függvényeknek hívjuk.

Az általános függvények összefoglalása

Az összes DOS-függvény a 33-as (hexa 21) megszakítással hívható. Az egyes függvényeket a függvényszám *AH*-regiszterbe helyezésével választjuk ki. A hagyományos DOS-függvények hívásai a 16.1. táblázatban bemutatott logikai csoportokba szervezettek. Azért, hogy ezeket az ábrákat a lehető legvilágosabbá tegyük, a DOS Technical Reference kézikönyvtől kissé eltérően rendezve írtuk le a függvényhívásokat. (A 16.2. táblázat sorolja fel az egyes függvényhívásokat.)

Mielőtt részletesebben is ismertetnénk ezeket a függvényeket, fel kell hívni a figyelmet, hogy e függvények némelyikének néhány tervezési és szervezési szempontja – különösen az 1-től 12-ig terjedőké – meglehetősen excentrikus. Ennek történeti okai vannak. A DOS számos részletét – különösen a DOS függvényhívásainak részleteit – úgy tervezték, hogy minél jobban utánozzák a CP/M által nyújtott szolgáltatásokat. Ez a fontos változtatás előzetes tervezési szempont volt, ami a lehető legkönnyebbé tette a 8 bites CP/M szoftver konvertálását a 16 bites IBM PC-re és DOS-ra. Bár a DOS megalkotásakor nagyszerű alkalom kínálkozott a múlt hibáitól megszabadulni, mégsem tettek így. (Talán nem túl szerencsés, de annál bölcsebb volt ez a döntés.) A DOS-szolgáltatások tiszta (vagy tisztább) újratervezése valójában a 2.00 változat fejlesztésekor indult meg, és a kiterjesztett függvényekben valósult meg. (☛ L. a 17. fejezetet.)

Az ezután következő oldalakon az összes DOS-változatban általánosan használt 46, eredeti DOS-függvény hívását részletezzük.

16.1. táblázat
A hagyományos DOS-függvényhívások logikai csoportjai

Függvény		Csoport
Dec	Hexa	
0	00	Nem berendezést kezelő függvény
1 – 12	01 – 0C	Karakteres I/O
13 – 36	0D – 24	Állománykezelés
37 – 38	25 – 26	További nem berendezést kezelő függvények
39 – 41	27 – 29	További állománykezelés
42 – 46	2A – 2E	További nem berendezést kezelő függvények

16.2. táblázat

A 33-as megszakítással kiváltott és az AH-regiszterben kiválasztott hagyományos DOS-függvényhívások

Függvény		Leírás
Dec	Hexa	
0	0	Befejezés: a program vége
1	1	Billentyűzetbemenet echóval
2	2	Monitorkimenet
3	3	Soros bemenet
4	4	Soros kimenet
5	5	Nyomtatókimenet
6	6	Közvetlen billentyűzet és monitor I/O
7	7	Közvetlen billentyűzetbemenet echó nélkül
8	8	Billentyűzetbemenet echó nélkül
9	9	Monitorstring
10	A	Tárolt billentyűzetbemenet
11	B	A billentyűzetbemenet állapotának ellenőrzése
12	C	„Töröld a billentyűzetet és hajtsd végre" funkció
13	D	A lemezegység alaphelyzetreállítása
14	E	Az aktuális lemezegység kiválasztása
15	F	Az állomány megnyitása
16	10	Az állomány lezárása
17	11	Az első illeszkedő adatállomány keresése
18	12	A következő illeszkedő adatállomány keresése
19	13	Adatállomány törlése
20	14	Soros állomány rekordjának olvasása
21	15	Soros állomány rekordjának írása
22	16	Állomány létrehozása
23	17	Az állomány átnevezése
24	18	A DOS-on belül használt függvény
25	19	Az aktuális lemezegység visszajelzése
26	1A	A lemezegység transzferterületének beállítása
27	1B	A FAT-információ lekérdezése az aktuális lemezegységről
28	1C	A FAT-információ lekérdezése bármely lemezegységről
29	1D	A DOS-on belül használt függvény
30	1E	A DOS-on belül használt függvény
31	1F	A DOS-on belül használt függvény
32	2O	A DOS-on belül használt függvény
33	21	A random állomány rekordjának olvasása
34	22	A random állomány rekordjának írása
35	23	Az állomány méretének lekérdezése
36	24	A randomadatmezők beállítása
37	25	A megszakításvektor beállítása
38	26	A programszegmens előállítása
39	27	A random állomány rekordjainak olvasása
40	28	A random állomány rekordjainak írása
41	29	Az állománynév elemzése
42	2A	A dátum lekérdezése
43	2B	A dátum beállítása
44	2C	Az idő lekérdezése
45	2D	Az idő beállítása
46	2E	A lemezegységre írás ellenőrzésének beállítása

A 0-s függvény: a program befejezése

A DOS 0-ás függvényével befejezhetjük a programot, és visszaadhatjuk a vezérlést a DOS-nak. Funkcionálisan a 15. fejezetben tárgyalt 32-es (hexa 20) DOS-megszakítással azonos. A kettő közül bármelyiket lehet használni a programból való kilépésre.

A 2.00 és az annál magasabb DOS-változatok a 76-os (hexa 4C) függvénnyel fejlettebb befejezésszolgáltatást nyújtanak, egy visszaadott kódot (hibakódot) hagyva az *AL*-regiszterben a program végén. Kötegelt feldolgozás esetén a DOS **ERRORLEVEL** utasítása használhatja ezt a kódot. A 0-ás függvény helyett a 76-os függvényt is használhatjuk, ha a program befejezésekor bármilyen hibát rögzíteni kívánunk. (☛ L. a 17. fejezetet.)

A 32-es DOS-megszakításhoz hasonlóan ez a függvény sem zárja le az állományokat automatikusan, amikor a program befejeződik. Akkor lehetünk biztosak, hogy a megváltoztatott állomány megfelelő hosszúságát rögzítjük az állománymutatóban, ha e függvény hívása előtt a 16-os vagy a 62-es állománylezárás-függvényt használjuk. A programnak kilépés előtt meg kell bizonyosodnia, hogy a PSP-cím valóban a *CS*-regiszterben van-e. A 15. fejezetben tárgyaltakból visszaidézhetjük, hogy a PSP tartalmazza a befejezési címet, és közli a DOS-szal, hová ugorjon, amikor a program befejeződik.

Az 1-es függvény: billentyűzetbemenet (echóval)

Az 1-es függvény a szabvány bemeneti berendezésről egy karaktert vár, és átadja azt az *AL*-regiszterben, amikor az rendelkezésre áll. Ezt a billentyűzet többi függvényhívásával is össze kell vetni, különösen a 6-os, a 7-es és a 8-as függvényhívással.

Íme az 1-es függvény működése! Az ASCII-karaktereket létrehozó billentyűzetműveletek egy bájtot adnak vissza az *AL*-ben, és ezeket azonnal jelzi ez a szolgáltatás. A 97-es speciális billentyűzetművelet, amely nem-ASCII-karaktereket ad vissza (☛ l. a 6. fejezetet), két bájtot generál, amelyet e szolgáltatásra vonatkozó, egymást követő két hívással kaphatunk meg.

A szolgáltatás használatának szabványos módja az *AL*-beli 0 vizsgálata. Ha az *AL* nem 0, akkor ASCII-karakterünk, ha az $AL = 0$, akkor pedig különleges karakterünk van (amelyet el kell menteni). A függvényhívást azonnal meg kell ismételni, hogy megkapjuk a pszeudo letapogatási kódot (scan code). (☛ A műveletek listáját, a kódokat és jelentésüket l. a 6. fejezetben.) Ugyanúgy, mint a DOS összes billentyűzetbemeneti szolgáltatásnál, a letapogatási kód az ASCII-karakterekre nem áll rendelkezésünkre, bár a ROM-BIOS-szolgáltatások kiegészítő bájtjában elérhetjük.

A DOS különböző billentyűzetszolgáltatás-függvényeit három kritérium szerint különböztetjük meg: várnak-e bemenetre, vagy jelzik, ha nincs felhasználható karakter; echózzák-e a bemenetet a képernyőn; ill. meg lehet-e szakítani a szolgáltatást a szabványos megszakításbillentyűvel. (Emlékeztetőül: a megszakításbillentyű, a *Ctrl-Break* vagy a *Ctrl-C* leütésére a szabványos DOS-művelet válaszol.) Van azonban olyan DOS-művelet is, amelyik nem engedélyezi a megszakításbillentyűre a válaszadást. A 2.00 változattal a DOS mégis bevezette a **BREAK ON**-utasítást, amely a DOS-nak olyan hatáskört ad, hogy az bármilyen körülmények között válaszolhasson a megszakításbillentyűre.

Az 1-es függvény mind a három műveletet végrehajtja: vár a bemenetre, echózza azt a képernyőre, és ha megszakításbillentyű-műveletet fedez fel, végrehajtja a 35-ös (cím) megszakítást.

🕮 MEGJEGYZÉS: A 2.00 és az annál magasabb DOS-változatoknál ez a billentyűzetszolgáltatás valójában a DOS szabványos bemeneti berendezéshez kapcsolt. Alapértelmezésben ez a billentyűzet, de ezt más bemeneti berendezésre át lehet irányítani. Ha a bemeneten nincs

beolvasható karakter, és várakozás nélkül használni kívánjuk ezt a szolgáltatást, vegyük figyelembe a 11. függvényt, amely ugyancsak azt jelzi, hogy a bemenet rendelkezésünkre áll-e vagy sem. (☞ E szolgáltatás változatait l. a 8-as és a 12-es függvénynél.)

A 2-es függvény: monitorkimenet

A 2-es függvény egy ASCII-karaktert ír a képernyőre (vagy a 2.00, vagy annál későbbi DOS-változatnál, a szabványos kimeneti berendezésre, amely más is lehet, mint a képernyő). A karatert *DL*-be kell helyezni.

Általában ez a szolgáltatás hat az ASCII vezérlőkaraktereire, éppúgy, mint a visszalépésre (*backspace*) vagy a kocsivisszára. A backspace karakternél a képernyőn a kurzort az aktuális sor eleje felé (visszafelé) mozgatjuk. A DOS Technical Reference kézikönyvben megadott információval (és a legelső óta kiadott összes DOS-kézikönyvvel) ellentétben nem íródik ki a kurzor elmozdulása után olyan üres karakter (hexa 20), amely hatékonyan törölne bármilyen előző karaktert. Ehelyett érintetlen marad minden információ, amely felett elhalad a kurzor.

A 3-as függvény: soros bemenet

A 3. függvény egy karaktert olvas be az *AL*-be a szabványos kiegészítő bemeneti berendezésről, amelyet szokásosan AUX-nak vagy COM1-nek ismerünk. A DOS **MODE**-utasítás varázslatával a beállítást megváltoztathatjuk úgy, hogy a bemenetet más berendezésről vegyük, pl. a COM2-ről. Szokásos esetben e bemenet az első RS – 232 soros port.

🕮 MEGJEGYZÉS: Ez a szolgáltatás bemenetre vár. Nem jelzi annak a számtalan problémának az állapotinformációját, amellyel a soros port küszködhet. Ha a soros port állapotát akarjuk tudni, használjuk a ROM-BIOS soros szolgáltatásait. (☞ L. a 12. fejezetet.)

A 4-es függvény: soros kimenet

A 4-es függvény a *DL*-regiszterből egy karakterkimenetet küld a szabványos kiegészítő berendezésre. (☞ L. a 3-as függvényt.)

5-ös függvény: nyomtatókimenet

Az 5-ös függvény a *DL*-ből egy bájtot ír ki a szabványos nyomtatóberendezésre, amelyet szokásosan PRN:-ként vagy LPT1:-ként ismerünk. (Ez a DOS **MODE**-utasításával lehet más berendezés is.) Átirányítás hiányában a szabványos nyomtató mindig az első párhuzamos nyomtatóadapter lesz, még akkor is, ha a soros portot használjuk nyomtatókimenetnek.

A 6-os függvény: közvetlen billentyűzet (monitor I/O)

A 6-os függvény egy összetett és excentrikus szolgáltatás, amely a billentyűzetbemenet és a képernyőkimenet műveleteit kombinálja egy rendezetlen csomaggá. Ugyanúgy, mint minden más esetben, a 2.00 és az annál későbbi DOS-változatokban az I/O nem a billentyűzethez és a

monitorhoz kapcsolódik, hanem inkább a szabványos bemeneti és kimeneti berendezésekhez (amely alapértékként a billentyűzet és a képernyő).

Lássuk, hogyan is működik a szolgáltatás! Az *AL*-regisztert használja bemenetnek, a *DL*-regisztert pedig kimenetnek. Ha a $DL=255$ (hexa FF), akkor az *AL*-ben akarunk egy bemeneti karaktert kapni. A zéro jelzőbit (flag, *ZF*) jelzi, hogy a bemenet kész. Ha a $ZF=1$, a bemenet még nem kész; ha $ZF=0$, akkor egy bemeneti bájt került az *AL*-be; ha pedig a *DL* nem 255, akkor feltételezzük, hogy egy helyes kimeneti karaktert tartalmaz, amelyet a szabványos kimenetre íratunk ki.

A 6-os függvény nem várja meg, hogy a billentyűzeten leüssünk valamit, a beolvasott karaktert nem hozza le, és nem szakítható meg megszakításbillentyűvel. (☛ L. még az 1-es függvényt.)

A 7-es függvény: közvetlen billentyűzetbemenet (echo nélkül)

A 7-es függvény a szabványos bemeneti berendezésről egy karakterre vár, és visszaadja azt az *AL*-regiszterben, amikor az rendelkezésünkre áll. Nem echózza a bemenetet a képernyőre, és nem engedi a megszakításbillentyű-műveletet érvényre jutni.

A 7-es függvény hasonlóképpen működik, mint az 1-es: az ASCII-karaktert egy bájtként adja vissza az *AL*-be, és azonnal jelenti is azt e szolgáltatással. A 97-es különleges billentyűzetművelet, amely nem-ASCII-karaktert ad vissza (☛ l. a 6. fejezet), hanem két bájtot generál. Ezeket a szolgáltatásra vonatkozó két egymást követő hívással kapjuk meg. E szolgáltatás szokásos használata az *AL*-beli 0 vizsgálat. Ha az *AL* nem 0, akkor ASCII-karakterünk, ha viszont az $AL=0$, akkor különleges karakterünk van. Ezt a karaktert el kell menteni, és azonnal meg kell ismételni a függvényhívást, hogy megkapjuk a pszeudo letapogatási kódot. Ez képviseli majd a speciális billentyűzetműveletet. (☛ A műveletek listáját, a kódokat és jelentésüket l. a 6. fejezetben). Ugyanúgy, mint a DOS összes billentyűzetbemeneti szolgáltatásánál az ASCII-karakterek letapogatási kódja nem áll rendelkezésünkre, bár a ROM-BIOS-szolgáltatásoknál ez elérhető a kiegészítő bájtban. (☛ L. a 6. fejezetet).

Hasonlítsuk össze az 1-es, a 6-os és a 8-as szolgáltatásokkal! Ha ezt a szolgáltatást várakozás nélkül akarjuk használni, amíg a bemenet nem áll rendelkezésünkre, alkalmazzuk a 11-es függvényt, amely visszajelzi majd, hogy van-e a bemeneten kész karakter vagy nincs. (☛ E szolgáltatás változatát l. a 12-es függvénynél.)

A 8-as függvény: billentyűzetbemenet (echo nélkül)

A 8-as függvény bemenetre vár, nem echózza azt, nem tiltja le a megszakításbillentyű kezelését. Azonos az 1-es függvénnyel, kivéve azt, hogy nem echózza a bemenetet a képernyőre (vagy a szabványos kimeneti berendezésre).

Részletesebben az 1-es függvénynél tárgyaltuk. Hasonlítsuk össze e szolgáltatást az 1-es, a 6-os és a 7-es függvénnyel! Ha várakozás nélkül akarjuk ezt a szolgáltatást használni, amíg a bemenet nincs kész, nézzük meg a 11-es függvényt, amely visszajelzi majd, hogy a bemenet készen van-e már. (☛ E függvény változatát l. a 12-es függvénynél.)

A 9-es függvény: string kiírása

A 9-es függvény egy karakterstringet küld a szabványos kimeneti berendezésre (amely alapértelmezésben a képernyő). A *DS* : *DX* regiszterpár adja a string címét. A $ karaktert **CHR$(36)** használjuk a string végének jelzésére.

🕮 MEGJEGYZÉS: Erre a függvényre – tévesen – mint „nyomtató string"-re hivatkoznak a DOS-irodalomban. Helyesebb volna, ha „képernyőstring"-ként képzelnénk el, mint ahogy itt is nevezzük.

Ez a szolgáltatás ugyan lényegesen kényelmesebb lehet, mint a bájtonkénti monitorszolgáltatások (a 2-es és a 6-os függvény), de hiányossága egy kijelezhető valós karakter, a $ használata a string lezárásaként. Ez azonban nem a legutóbbi idők hibája, hanem egy másik CP/M kompatibilitás mellékterméke. Ha nem biztos, hogy a kimenetre $ jelet nem akarunk írni, el kell kerülnünk ezt a szolgáltatást. Igaz, a kiegészített DOS-függvények (☛ l. a 17. fejezetet), a **CHR$(0)**-t használják a string lezárására. Ez a gyakorlat a UNIX operációs rendszer és a nyelv konvencióit követi.

A 10-es függvény (hexa A): tárolt billentyűzetbemenet

A 10-es függvény egy csodálatos szolgáltatás: munkába állítja programjainkban a DOS szerkesztőbillentyűit. A szolgáltatás bemenete egy string, amelyet nem karakterenként, hanem feldolgozási egységként kezel. Ha a billentyűzeten éppen begépelt karakterek alkotják a bemenetet (nem pedig valahonnan máshonnan átirányítással olvasunk), a begépelő használhatja a DOS összes szerkesztőbillentyűjét. Ha megnyomjuk a *Return*-billentyűt (vagy egy kocsivissza (**CHR$(13)**) kerül sorra a bemeneti adatállományban), a bemeneti művelet befejeződik, és a teljes string bekerül a programunkba. E szolgáltatás használata számtalan előnnyel jár, különösen akkor, ha a programjainknak teljesen összefüggő billentyűzetbemeneti stringekre, nem pedig bájtonkénti bemenetre van szükségünk. A legfontosabb előnye, hogy megkímélhetjük magunkat a részletes bemenetkezelő kódok írásának erőfeszítéseitől, és programjaink használóinak egy ismert bemenetszerkesztési segédletet adhatunk: a DOS szövegszerkesztési konvenciót.

E szolgáltatás használatához a DOS-t el kell látnunk egy bemeneti tárolóterülettel, ahol a bemeneti string felépülhet. A *DS*:*DX* regiszterpár mutat erre a tárolóterületre (puffer). A tároló első bájtjában tájékoztatjuk a DOS-t a tároló méretéről, ez tulajdonképpen a bájtok száma, amelyet a DOS bemeneteként használhatunk. A tároló második bájtja jelzi a bevitt bájtok tényleges számát. A DOS a bemeneti stringet, amely teljes egészében ASCII-karaktereket tartalmaz, a 3. bájttal kezdődően helyezi a tárolóba. A bemeneti string végét a kocsivissza karakter, **CHR$(13)** jelzi. A kocsivissza karaktert is beírjuk a pufferbe, számára tehát helyet kell fenntartani, de a DOS a visszaadott karakterek számába (második bájt) nem számolja bele.

Eszerint a DOS által adható leghosszabb tároló 255 munkabájt, a visszaadható leghosszabb string pedig eggyel kevesebb, mint a munkahossz. Mivel a tároló első két bájtját állapotinformációra használjuk, tényleges munkamérete két bájttal kevesebb a tároló teljes méreténél. Ezzel magyarázhatjuk a DOS és a BASIC bemenetének nehézkes konvencióit.

Ha a bemenet a A DOS által tárolóba helyezhető méret után folytatódik (ami egy bájttal rövidebb, mint a munkahossz), akkor a DOS figyelmen kívül hagyja bármelyik ezt követő bemenetet; mindaddig sípol, amíg nem találkozik egy kocsivisszával.

A függvény néhány elemét ellenőrizhetjük a DOS-utasításértelmező (command interpreter) által elfogadott bemenő karakterek megszámlálásával. Egyszerű kísérletek feltárják, hogy

az utasításértelmező 128 bájt méretű munkatárolót használ (a teljes hossz 130 bájt). A DOS tiltakozni fog bármely kocsivisszától eltérő bemenet ellen a 127. bájt után. (☛ E szolgáltatás változatát l. a 12-es függvénynél.)

A 11-es függvény (hexa B): a billentyűzetbemenet állapotának ellenőrzése

A 11-es függvény jelzi, hogy a billentyűzetről (vagy a szabványos bemeneti berendezésről) a bemenet készen áll-e. Ez különösen hasznos a nemvárakozó billentyűzetbemeneti szolgáltatásoknál (l. az 1-es, a 7-es és a 8-as függvényt). Ha a bemenet kész, akkor $AL=255$ (hexa FF); ha nem kész, akkor $AL=0$.

A szabványos megszakításbillentyű-művelet erre a szolgáltatásra aktív. (☛ L. a magyarázatot az 1. függvénynél.)

A 12-es függvény (hexa C): töröld a billentyűzetet, és hajtsd végre!

A 12-es függvény törli a RAM-ban a billentyűzettárolót, és a következő DOS-szolgáltatások egyikét hajtja végre: az 1-es, a 6-os, a 7-es, a 8-as vagy a 10-es függvényt. Az *AL*-regiszterrel választjuk ki, hogy e függvények közül melyiket fogjuk végrehajtani, miután a billentyűzettároló törlődött. Az oda nem tartozó karakterektől megszabadulva, ez a függvény a rendszert arra készteti, hogy új bemenetre várjon, mielőtt a kiválasztott függvényre válaszolna.

Mivel a 6-os függvényt támogatja, a következő szolgáltatásnak nem kell billentyűzetbemenetnek lennie, lehet képernyőreírás is.

A 13-as függvény (hexa D): a lemezegység alapállapotba állítása

A 13-as függvény alaphelyzetbe állítja a lemezegységet, és kiüríti az összes állomány átmeneti tárolóit. Úgy biztosíthatjuk, hogy a katalógusban (directory) a megváltoztatott állomány valós hossza rögzítődjön, ha e funkció hívása előtt a 6-os vagy a 32-es állománylezáró függvényt használjuk. A korábbi DOS-kézikönyvekben leírtakkal ellentétben, ez a szolgáltatás nem állítja vissza a lemezegység alapértékét A-ra (vagy nem végez más hasonló változtatást), még azokban a DOS-változatokban sem, amelyek ezt állítják magukról.

A 14-es függvény (hexa E): az aktuális lemezegység kiválasztása

A 14-es függvény kiválasztja az aktuális alaplemezegységet, és jelenti az installált lemezegységek számát. A lemezegységet a *DL*-ben választjuk ki, az *A* lemezegységet 0-val, a *B*-t 1-gyel és így tovább. A lemezegységek számát az *AL*-ben adja vissza. Az alapegység azonosítószáma az installált egységek számánál eggyel kisebb szám és a nulla közé eső egész szám lehet. A szolgáltatás használatakor figyelnünk kell néhány dologra. Soha nincs köz a DOS által használt lemezegység-azonosítók között; egymást követő sorrendben vannak számozva. Ha

csak egy fizikai lemezegység van, a DOS az 1. lemezegységet (*B* lemezegység) szimulálja. A lemezegységet azonosító betűt pedig úgy kapjuk meg, hogy a lemezegység számát hozzáadjuk az A karakterhez (**CHR$(65)**). Az a valószínűtlen eset, amikor 26-nál több lemezegység van, néhány igazán furcsa lemezegység-„betűt" eredményezhet.

A 25-ös (hexa 19) függvény visszaadja az aktuális lemezegység számát; a 14-es és a 15-ös függvényt pedig kombinálhatjuk, hogy az aktuális alaplemezegység beállításának zavarása nélkül megtudjuk a lemezegységek számát. Ezt assembly nyelven a következőképpen tehetjük:

```
MOV    AH,25    : az aktuális lemezegységet jelzi
INT    33       : függvényhívás
MOV    DL,AL    : az aktuális lemezegység számát átmásolja
MOV    AH,14    : az aktuális lemezegységet beállítja
INT    33       : függvényhívás
```

Ennél a pontnál az *AL* a lemezegységek számát tartalmazza, de nem változtatja meg az aktuális lemezegységet. Annak érdekében, hogy a számot a legmagasabb lemezegység betűjére konvertáljuk, hozzáadhatjuk a következő utasítást:

```
ADD    AL,'A'-1
```

A 15-ös függvény (hexa F): állomány megnyitása

A 15-ös függvény az állományt hagyományos DOS-módon, az állományvezérlő tömb (file control block, FCB) használatával nyitja meg. Az FCB az állományon belül egy 128 logikai rekordból álló csoport, a nyitott állományra vonatkozó információkészletet tartalmazza. Ezt a DOS arra használja, hogy megtalálja az állományt és az állomány adatait. (☛ Az FCB részleteit e fejezet végén tárgyaljuk.)

Programjaink a *DS*:*DX* regiszterpár révén mutatnak az FCB-re. A DOS megkísérli az FCB-ben megadott specifikációjú állomány megnyitását. Az eredményt az *AL*-ben jelzi. Az *AL* = 255 (hexa FF), ha a nyitás sikertelen; ha sikeres, akkor az *AL* 0-ra állítódik. Ne feledjük: csak a már létező állományokat lehet megnyitni! (☛ L. a 22-es függvényt, az állománylétrehozás szolgáltatást.) Rendszerint akkor használjuk ezt a függvényt, ha bemeneti állományt nyitunk meg, a 22-es függvényt pedig akkor, amikor egy kimeneti adatállományt.

Sikeres megnyitáskor a DOS az FCB-ben számos mezőt beállít. Ha a 14-es függvénnyel meghatározzuk az alaplemezegységet, a DOS betölti a ténylegesen használt lemezegységet. Ez a konvenciótól eltérő lemezegység-azonosító számmal történik. Ekkor (ellentétben a legtöbb lemezegység-műveletnél használttal) az A lemezegységet 0 helyett 1-gyel jelöljük. A DOS betölti az állomány dátumát és időpontját, és beállítja az aktuális tömböt 0-ra.

Valami rendkívüli megy végbe az FCB rekordméretmezővel, amely azt a rekordméretet mutatja, amit az állományolvasásnál és -írásnál használni kívánunk.

Ezt a mezőt rendszerint azután adjuk meg, hogy az állományt megnyitottuk. Ennek ellenére az állomány nyitásakor a DOS ezt a mezőt egy alap-rekordméretre, 128-ra (hexa 80) állítja be. Használhatjuk ezt a rekordméretet is, de felhasználásunktól függően meg is változtathatjuk. Számos szövegszerkesztő program a 128 bájtos rekordméretet használja hatékony I/O műveletekhez. Érdekes módon a DOS editora az Edlin, nem ezt alkalmazza. (☛ A rekordméretmezőre vonatkozó további információkat l. az FCB részletes ismertetésében.)

A 16-os függvény (hexa 10): állomány lezárása

A 16-os függvény lezárja az állományt, amelynek FCB-jére a *DS* : *DX* regiszterpár mutat. Ha sikeres a művelet, akkor az $AL=0$, ha hibás, az $AL=255$ (hexa FF).

Az adatállományt le kell zárni bármely katalógus aktualizálására irányuló művelet után. A DOS a katalógusban levő lemezegység-azonosító és a nyitott FCB-ben levő azonosító összehasonlításával felismeri, hogy ugyanazon a lemezen ugyanazt az adatállományt zárjuk le, amelyet megnyitottunk. Ez némi védelmet nyújt a lemezes információk összekeverése ellen, amely akkor léphet fel, amikor a felhasználó lemezt cserél, és erre az új lemezre akar írni, mielőtt a régi állományt lezárta volna.

A 17-es függvény (hexa 11): az első illeszkedő állomány keresése

Ha több névhez is illeszkedő specifikációval akarunk kikeresni néhány (vagy az összes) állománynevet, a kezdő lépés a 17-es függvény meghívása. A *DS*:*DX* regiszterpár a keresendő állomány nevét tartalmazó FCB-re mutat. Ennek a szolgáltatásnak a használatakor olyan állománynév meghatározásokat is kezelni tudunk, amelyek a "?" és a "*" globális karaktereket foglalják magukba. Ez a szolgáltatás elkezdi a folyamatot, a 18. függvény pedig folytatja a keresést a többi állományra.

Az *AL* hibát (255, hexa FF) vagy sikert (0) jelez. Ha ez a szolgáltatás sikeres, a DOS az állománynevet az FCB állománynévmezőbe másolja, ezzel az állományt megnyitásra késszé teszi. Az FCB a 18-as függvény "Találd meg a következőt" szolgáltatásával való folytatáshoz szükséges információt is tartalmazza.

✱ FIGYELMEZTETÉS: *Bármely más művelet az ezen FCB-t használó állományon tönkreteszi a "Találd meg a következőt" szolgáltatáshoz szükséges információt. Ha több állományt is ki akarunk keresni, hogy rajtuk I/O műveletet végezzünk, akkor az FCB-ben levő keresési információt ki kell mentenünk!*

Ha az FCB-nek van kiterjesztése (☛ l. az FCB részletes ismertetését), akkor a keresendő állomány attribútumait is megnevezhetjük. Az attribútum keresése különös logikát követ. Ha a rejtett (rendszer vagy tárgymutató) attribútumbitek bármely kombinációját is meghatározzuk, a keresés a rendes állományok – és bármely más állomány – attribútumait összehasonlítja ezekkel az attribútumokkal. Ha a kötetcímke-attribútumot (volume label attribute) határozzuk meg, akkor a függvény a tárgymutatóban csak a belépési pontokat nézi át ezen attribútum szerint. A 2.00 előtti DOS-változatoknál sem a tárgymutató, sem a kötetcímke-attribútumokat nem lehetett használni az állomány keresési műveletre. Az archív (archive) és a csak olvasható (read-only) attribútumokat nem lehet használni keresési kritériumként egyik DOS-változatban sem.

Előnyös ez a szolgáltatás akkor is, ha a programjainkkal nem akarunk több állománnyal dolgozni. Ha az első illeszkedő állomány az egyetlen, ezt megtalálhatjuk e szolgáltatással, természetesen az állomány megnyitása előtt. Programunkat alkalmassá tehetjük arra, hogy globálisan állományneveket is elfogadjon. A következőkben bemutatjuk azt az elvet, amelyet a BASIC-ben különböző állományok keresésekor és az azokon való műveletek végrehajtásakor használhatunk:

```
10 ' A 17-es függvény meghívása
20 WHILE FILE FOUND
```

```
30  SAVE FCB FOR FIND-NEXT
40 ' Végezd el azt, amire szükség van, hogy kimentsd az FCB mezőt!
50  RESTORE FCB FOR FIND-NEXT
60 ' A 18-as függvény meghívása
70  WEND
```

A 18-as függvény (hexa 12): a következő illeszkedő állomány keresése

A 18-as függvény találja meg az állományok sorából a következőt, követve a 17-es függvény által elvégzett előkészítési beállításokat. (☛ A beállítási műveletre, a visszaaadott kódokra és a speciális figyelmeztetésekre vonatkozó részleteket l. a 17-es függvénynél.)

A 19-es függvény (hexa 13): állomány törlése

A 19-es függvény törli a *DS* : *DX* regiszterpár által mutatott FCB-hez illeszkedő állományokat. Az $AL=0$, ha a művelet sikeres, és az összes illeszkedő tárgymutató belépési pontjai törlődtek. Az $AL=255$ (hexa FF), ha a művelet sikertelen, azaz nem volt illeszkedő tárgymutató-bevitel. (☛ Az FCB részleteiről bővebben a későbbiekben lesz szó.)

A 20-as függvény (hexa 14): soros állomány rekordjának olvasása

A 20-as függvényt használjuk az állományban a következő soros rekord kiolvasására. E függvény hívása előtt a *DS* : *DX* párt kell használnunk arra, hogy az állomány FCB-jére mutassunk. A soros rekord számát az FCB aktuális tömb és aktuális rekordmező értékeiből vesszük. Ezután az adatállományt kiolvassuk, és sikeres vagy részlegesen sikeres műveletnél az adat az aktuális lemez transzferterületre kerül. (☛ L. a 26. függvényt.)

A DOS eggyel megnöveli az FCB rekord címmezőket minden egyes olvasás után, hogy elősegítse az adatállomány soros olvasását. Mi magunk is megváltoztathatjuk a címmezőket, ha az állományban lépkedni akarunk. A soros állományban ez viszont már nem túl célszerű; leghelyesebb a 33-as és a 34-es random I/O függvényeket használni, ha az állomány rekordjaihoz random hozzáférésre van szükség. (☛ A random rekordszámok és a soros rekordszámok számontartási metodikájának eltérésére l. az FCB részletezésénél lévő megjegyzéseket.)

Az *AL*-t használjuk az olvasás eredményének jelzésére. A sikeres olvasást jelzi, ha az $AL=0$. Az $AL=1$ jelzi az állomány végét, mutatva, hogy nem olvastunk adatot; az $AL=2$ jelzi, hogy adatot lehetett beolvasni, de a DTA-ban (lemezegység transzferterület) nincs elég hely a teljes rekord számára. Az $AL=3$ jelzi az állomány végét részleges rekordolvasással (a rekord 0 értékű bájtokkal töltődik ki).

A 21-es függvény (hexa 15): soros állomány rekordjának írása

A 21-es függvény soros rekordokat ír, és az előző szolgáltatás, a 20-as függvény társa. A *DS : DX* regiszterek az FCB-re mutatnak, ahol a rekord címét tároljuk. A cím kiolvasása után a DOS a DTA-ból veszi az adatokat, és azokat a lemezre írja. (☛ L. a 26-os függvényt.)

Ha a szolgáltatás befejeződött, az *AL* a következő visszaadott kódértékek egyikét tartalmazza: az $AL=0$ jelzi a sikeres végrehajtást; az $AL=1$ jelzi, hogy a lemez megtelt; az $AL=2$ pedig azt, hogy nincs elég terület a lemez transzferszegmensén a rekordíráshoz. Természetesen a DOS belső lemeztranszfer-szegmensének elég hellyel kell rendelkeznie ahhoz, hogy a rekordokat a lemez transzferterületünkre elhelyezze.

Ez a szolgáltatás az adatokat csak logikailag írja, de nem szükségszerűen fizikailag is. A DOS tárolja a kimeneti adatokat mindaddig, amíg egy komplett szektorra való össze nem gyűlik. Ekkor írja ki a szektort. Mindaddig, amíg egy állományt le nem zárunk, lehetnek adatok a DOS tárolójában. Ezekre a programunk úgy tekint, mintha már a lemezre kiírtak lennének, noha ezeket még nem vittük át a lemezre. Ez nehézséget jelenthet, ha a programot rendellenesen fejezzük be.

A 22-es függvény (hexa 16): állomány létrehozása

A 22-es függvény katalógus belépési pontot (directory entry) talál, vagy azt hoz létre. A szolgáltatás először illeszkedő állománynevet keres a katalógusban, ha ilyet nem talál, akkor üres belépési pontot keres. Ezután megnyitja az állományt, ha adott egy FCB-mutató a *DS : DX*-ben. Ezt a függvényt rendszerint a kimeneti, a 15-ös függvényt pedig a bemeneti állományok megnyitására használjuk.

Az $AL=0$ jelzi, hogy a művelet sikeres, az $AL=255$ (hexa FF) pedig hibát jelez, amely általában a katalógus helyhiányának következménye. Amikor állományt nyitunk ezzel a függvénnyel, annak hossza 0-ra állítódik. Ha új információt írunk egy már létező állományba, és ezt a 16-os függvény meghívása követi, akkor az állomány régi tartalmát az új információval felülírjuk. Ez nem történhet meg, ha az állományt a 15-ös függvénnyel nyitjuk, mert az úgy állítja be az FCB állományméret-mezőt, hogy az illeszkedjék a katalógusban talált állománymérethez.

A 23-as függvény (hexa 17): állomány átnevezése

A 23-as függvény átnevezi az állományokat a módosított FCB-vel, amelyre a *DS : DX* mutat. Az átnevezést végző műveletek során az FCB-t különleges módon kezeljük. Bár a lemezegység és az eredeti állománynév a szokásos pozíción helyezkedik el, az új állománynevet 16-os eltolással helyezzük az FCB-be, szokásos esetben az állományméret számára félretett mezőktől kezdve. (☛ L. az FCB részletes ismertetését „Az állományvezérlő tömb" című részben.)

Az $AL=0$ a sikert jelzi, az $AL=255$ (hexa FF) pedig azt, hogy vagy nem találtunk újranevezendő állományokat, vagy az új állománynév már használatos.

Ha az új állománynév globális karaktereket, pl. *-ot vagy ?-et tartalmaz, azokat "Másold át a régi névből" jelnek értelmezzük, ezért nem változnak azok a karakterek, amelyek az eredeti névben a globális karaktereknek megfelelő helyen voltak.

A 24-es függvény (hexa 18): a DOS-on belül használt függvény

A 24-es függvényt és a 29 – 32-es függvényeket a DOS belső céljaira használja. Korlátozott számban ugyan e szolgáltatásokról is rendelkezünk információkkal, mégsem lenne bölcs dolog ezeket felhasználni vagy terjeszteni. Természetesen mi sem támaszkodhatunk a későbbi DOS-változatoknál azokra a DOS-függvényekre, amelyeket nem hirdetnek széles körben felhasználható szolgáltatásként.

A 25-ös függvény (hexa 19): az aktuális lemezegység visszajelzése

A 25-ös függvény jelzi az aktuális lemezegységet az *AL*-ben, a lemezegység szabványos számkódját használva. Az *A* lemezegység 0, a *B* lemezegység 1. (☛ E függvény assembly nyelvű használatára l. a 14-es függvénynél mutatott példát.)

A 26-os függvény (hexa 1A): a lemezegység transzferterületének beállítása

A 26-os függvény állítja be a lemezegység transzferterületét, amelyet a DOS az állomány I/O-ra használ. A DTA helyét a *DS : DX* regiszterpár határozza meg. A PSP-ben rendelkezésünkre áll egy 128 bájtos alap-DTA, hexa 80 eltolással.

A 27-es függvény (hexa 1B): FAT-információ lekérdezése az aktuális lemezegységről

A 27-es függvény adja vissza az aktuális lemezegységben lévő lemezre vonatkozó kulcsinformációt. A 28-as függvény azonos szolgáltatást végez bármely lemezegységre vonatkozóan. A következő fejezetben tárgyalt 54-es függvény is közel azonos szolgáltatást végez.

E függvényhívás a következő információt adja vissza: az *AL* tartalmazza a szektorok számát allokált egységenként (1 szektor az egyoldalas lemez, 2 szektor a kétoldalas esetén). A *CX* tartalmazza bájtokban a lemezszektorok méretét (a szabványos PC-formátum 512 bájt). A *DX* a lemezen allokált egységek (klaszterek) teljes számát tartalmazza. A *DS : BX* a DOS-munkaterületben a FAT-azonosítót tartalmazó bájtra mutat. (☛ A FAT-ról részletesebben l. az 5. fejezetben.) A 2.00 DOS-változat előtt a *DS : BX* regiszterpár a teljes lemez FAT-ra mutatott, erről biztosan tudtuk, hogy teljes egészében a memóriában van. Ennek első bájtja lett az azonosító bájt. A későbbi DOS-változatokban a FAT nem szükségszerűen van egészében egy helyen jelen, így biztonságosabb azt feltételezni, hogy a *DS : BX* csak egyetlen azonosító bájtra mutat.

Óvakodjunk egy veszélyes lehetőségtől! Ez a függvény ugyanis a *DS : BX*-et úgy állítja be, hogy az a FAT azonosítóbájtjára mutat, amely nincs az adatszegmens területen elhelyezve. Ez azt jelenti, hogy a függvény felülírja az adatszegmens-regiszter (*DS*) értékét. Rendszerint ez megzavarja azoknak a programoknak a működését, amelyek a *DS*-regiszter beállításokon alapulnak. Ezt a nehézséget a legcélszerűbben úgy kerülhetjük el, ha elmentjük, majd visszaállítjuk a *DS*-értéket. Az alábbiakban erre látunk egy példát.

```
PUSH    DS          ; a DS-cím mentése
MOV     AH,27       ; a függvény lekérdezése
INT     33          ; a DOS-függvényhívás kiváltása
MOV     AH, [BX]    ; a FAT azonosítóbájtjának kiragadása
POP     DS          ; a DS-cím visszahelyezése
```

Ez a kis probléma jó példa arra, hogyan lehet egy programtervezésben a legkönnyebben hibákat elkövetni. Ha ez a függvény a *DS* helyett az *ES*-regisztert használná, már nem is jelentene buktatót programjaink számára.

A 28-as függvény (hexa 1C): FAT-információ lekérdezése bármely lemezegységről

A 28-as függvény majdnem ugyanúgy működik, mint a 27-es, de nemcsak az aktuális lemezegységekről ad információt, hanem bármely más lemezegységről is. Mielőtt ezt a szolgáltatást hívnánk, állítsuk be a *DL*-t a speciális lemezegység-azonosító számára, ahol 0 = aktuális lemezegység, 1 = *A* lemezegység, 2 = *B* lemezegység stb. (Vegyük észre, hogy nem a konvencionális lemezegység-számozási eljárást használjuk, amely az *A* lemezegységet 0-ként, a *B* lemezegységet pedig 1-ként határozza meg.)

A 33-as függvény (hexa 21): random állomány rekordjának olvasása

A 33-as függvény az állományból egy random helyen levő rekordot olvas ki. A *DS*:*DX* mutasson az állomány FCB-jére, majd határozzuk meg azt a random rekordot, amelyet a FCB-ben levő random rekordmező beállításával ki akarunk olvasni. A rekordmező beállítása után kerül be az adat az aktuális DTA-ba.

Az *AL*-t ugyanazokkal a kódokkal állítjuk be, mint a soros olvasásnál: az *AL*=0 azt mutatja, hogy az olvasás sikeres. Az *AL*= 1 az állomány végét jelzi. Az *AL*=2 jelzi, hogy nincs elég terület a lemezátvitel szegmensében, az *AL*=3 pedig az adatállomány végét jelzi, de részleges adatrekord rendelkezésünkre áll.

🕮 MEGJEGYZÉS: Programunknak folyamatosan be kell állítania a random rekordmezőt. Ezzel ellentétben a DOS automatikusan eggyel megnöveli a soros mezőket, beállítva azokat arra, hogy a következő sorban levő rekordhoz illeszkedjenek. Emiatt sokszor kényelmesebb felváltva alkalmazni a két eljárást úgy, hogy egy random olvasást követünk soros olvasásokkal.

Vessük össze ezt a függvényt a 39-es függvénnyel, amely egy random rekordnál többet tud olvasni, vagy a 20-as függvénnyel, amely soros rekordokat olvas! (☛ A random rekordmezők beállítását részletesebben a 36-os függvénynél ismertetjük.)

A 34-es függvény (hexa 22): random állomány rekordjának írása

A 34-es függvény egy rekordot ír az állományban levő egyik random helyre. Mivel a *DS*:*DX* regiszterpár az állományra vonatkozó FCB-re mutat, programunknak be kell állítania a random rekordmezőt az FCB-ben, hogy megfeleljen annak a random rekordnak, amelyet írni akarunk. Miután beállítottuk a random rekordmezőt, az adat kiíródik a jelenlegi DTA-ból.

Az AL-t ugyanazokkal a kódokkal állítjuk be, mint a soros írásnál: a 0 a sikeres írást jelzi; az 1 azt, hogy a lemez tele van; a 2 pedig azt, hogy nem áll elegendő hely rendelkezésünkre a lemezátvitel szegmensében.

🕮 MEGJEGYZÉS: A programunknak folyamatosan be kell állítania a random rekordmezőt, hogy minden beírt random rekordnak megfeleljen: ezzel ellentétben a DOS automatikusan eggyel megnöveli a soros mezőket, beállítva azokat arra, hogy a következő sorban levő rekordhoz illeszkedjenek. Emiatt sokszor kényelmesebb felváltva alkalmazni a két eljárást: egy random írást követünk soros írásokkal.

Vessük össze ezt a függvényt a 40-es függvénnyel, amely több random rekordot tud írni, vagy a 21-es függvénnyel, amely soros rekordokat ír! (☛ A random rekordmező beállításáról további információt a 36-os függvénynél találhatunk.)

A 35-ös függvény (hexa 23): az állományméret lekérdezése

A 35-ös függvény az állomány méretét mint az állományban lévő rekordok számát jelzi. A $DS:DX$ annak az állománynak az FCB-jére mutat, amelyre kíváncsiak vagyunk. A függvény hívása előtt az FCB-nek lezártnak kell lennie, és a rekordméretmezőt az FCB-be kell töltenünk. Ha a rekordméretet 1-re állítjuk be az állományméretet bájtokban fogja jelezni – valószínűleg éppen erre van szükségünk.

Ha a művelet sikeres, az $AL=0$, és az állomány bekerül az FCB-be. Ha nem találunk állományt, az $AL=255$ (hexa FF).

A 36-os függvény (hexa 24): random rekordmezők beállítása

A 36-os függvény állítja be a random rekordmezőt, hogy az megfeleljen az aktuális soros tömbnek és az FCB-ben levő rekordmezőknek. Ez teszi lehetővé, hogy a sorosról a random I/O-ra váltsunk át. A $DS:DX$ regiszterek a nyitott állomány FCB-jére mutatnak.

A 37-es függvény (hexa 25): a megszakításvektor beállítása

A 37-es függvényt használjuk a megszakításvektor beállítására. A $DS:DX$ regiszterpár tartalmazza a megszakításkezelő szubrutinvektor címét, az AL pedig a megszakítás számát. A 4 bájtos megszakításvektort a vektortáblába helyezzük, és az **INT**-utasításban a megszakítás sorszámával hívjuk meg.

Bármely program, amely eleget tud ahhoz, hogy megszakításvektorokat hozzon létre, be tudná állítani saját maga is: ez a szolgáltatás viszont megszabadít minket néhány olyan trükk elvégzésétől, amelyre egy megszakításvektor biztonságos beállításához szükség van. Azoknál a vektoroknál, amelyek a megszakításkezelőkre mutatnak, szokásosan a szegmensrész az aktuális kódszegmens (CS) lenne, amelyet a DS-be kell átvinni. Vegyük észre, mennyire ügyetlen volt az ES helyett a DS-t választani, mert ez megbolygathatja a program felhasználóinak hozzáférését az adatokhoz. (☛ A megszakításvektor tartalmát l. a következő fejezetben az 53-as (hexa 35) függvénynél.)

A 38-as függvény (hexa 26): programszegmens létrehozása

A 38-as függvény egy külön tölthető alprogram vagy átlapoló programrész (overlay) végrehajtásának előkészítésére új programszegmenst hoz létre. A *DX*-et használjuk arra, hogy az új programhoz szegmensparagrafust adjon. Az aktuális program programszegmens-előtagját (PSP) az új szegmensterület első 256 (hexa 100) bájtjába másoljuk, ezzel új PSP-t hozunk létre. Ezt az új PSP-t aktualizáljuk az új memória- és megszakításvektor információkkal. Miután ez a szolgáltatás beállítja a PSP-t, a konvencionális DOS bemeneti szolgáltatásaival olvashatjuk be a COM típusú programállományokat közvetlenül a PSP-t követő területre. A DOS fejlettebb változatainál (a 2.00 és a későbbiek) a beállítás és az overlay használatának folyamata lényegesen könnyebb. (☛ Erről részletesebben a következő fejezetben, a 75-ös függvénynél olvashatunk. A programszegmens-előtag (PSP) magyarázatát l. a 15. fejezetben.)

A 39-es függvény (hexa 27): random állomány rekordjainak olvasása

A 33-as függvénytől eltérően, a 39-es függvény egy vagy több rekordot olvas be egy random állományba. A *DS*:*DX* az olvasandó állományra vonatkozó FCB-re mutat, és a random rekordszámot ekkor ebből az FCB-ből vesszük. A *CX* a kívánt rekordok számát tartalmazza, amelynek 0-nál többnek kell lennie.

A visszaadott kódok ugyanazok, mint a 33-as függvénynél. Az $AL=0$ jelzi azt, hogy az olvasás sikeres volt; az $AL=1$ pedig az állomány végét, további adat nélkül. (Ha a rekordokat beolvastuk, akkor az utolsó rekord teljes). Az $AL=2$ jelzi a lemezátvitel szegmenskezelési problémáit (gyakran a hexa FFFF-t meghaladó eltolódású szegmenseket, amelyek nem megengedettek). Az $AL=3$ a rekord végét jelzi (ha az utolsó rekord nem teljes, akkor kiegészíti nullákkal).

Függetlenül attól, hogy mi az eredmény, a *CX*-ben az olvasott rekordok száma található – beleértve a részleges rekordokat is –, és az FCB-ben a random rekordmező a következő random rekordra állítódik.

☛ Vessük ezt össze a 33-as függvénnyel, amely csak egy rekordot olvas be!

A 40-es függvény (hexa 28): random állomány rekordjainak írása

A 34-es függvénytől eltérően, a 40-es függvény egy vagy több rekordot ír egy random állományba. A *DS*:*DX* az írandó állományra vonatkozó FCB-re mutat, a random rekordszámot ebből az FCB-ből vesszük. A *CX* a kívánt rekordok számát tartalmazza, és ebben az esetben a *CX* 0 is lehet. A $CX=0$ jelzi a DOS számára, hogy a továbbiakban az állomány végének azt a helyet kell tekinteni, ahol az állományban most tart. Ez könnyebbé teszi programjaink számára a random állományokkal való gazdálkodást: ha logikai szinten töröltünk rekordokat egy állomány végéről, ez a szolgáltatás lehetővé teszi a programjainknak, hogy az állományról az e pont mögött lévő részeket levágjuk, és lemezterületet szabadítsunk fel.

A visszaadott kódok ugyanazok, mint a 34-es függvénynél: az $AL=0$ jelzi a sikeres írást, az $AL=1$ pedig azt, hogy nincs már több felhasználható lemezterület. Az eredménytől függetlenül a *CX*-ben mindig a lemezre írt rekordok száma található.

Vessük össze ezt a 34-es függvénnyel, amely csak egy random rekordot ír!

A 41-es függvény (hexa 29): állománynév elemzése

A 41-es függvény elemzi az állománynévre vonatkozó utasítássort a következő formával:

DRIVE:FILENAME.EXT.

Ha megtalálja az állománynevet, létrehoz egy FCB-t. A 41-es függvény különösen hasznos, ha a programnak fel kell dolgoznia a meghívásakor kapott állománynév-paramétereket. A paraméterek elemzését gondosan tervezték meg, hogy a program számára kényelmessé tegyék az alaplemezegység, az állománynév és a kiterjesztés beállítását. Ezeket utasítás specifikálással lehet felülbírálni, ami a program viselkedését jellemző néhány általános sajátosság alapját veti meg.

A *DS*:*SI* regiszterpár mutat az állományspecifikus stringre. Az *ES*:*DI* regiszter arra a memóriahelyre mutat, ahova a még meg nem nyitott FCB-t töltjük be. Az *AL* 0–3. bitjei vezérlik az állománynév elemzését.

Ha a **0. bit = 1**, a függvény a szeparátor utáni részt vizsgálja (pl. a bevezető üres helyeket), hogy megtalálja az állományleírást.

Ha a **0. bit = 0**, akkor a vizsgálatot nem hajtja végre, és az állományleírást úgy tekinti, hogy az az utasítássor első bájtjának helyén van.

Ha az **1. bit = 1**, akkor az FCB-ben a lemezegység bájtja csak akkor állítódik be, ha a vizsgált állományban ez kifejezetten szerepel. Ez lehetővé teszi az FCB-nek, hogy inkább saját alaplemezegységével rendelkezzen (amit felül lehet bírálni), mint hogy a DOS alaplemezegységét használja.

Ha a **2. bit = 1**, az FCB-ben levő állománynév csak akkor változik, ha az állomány leírásban érvényes állománynév található. Ez lehetővé teszi programjainknak, hogy alap-állományneveket állítsanak be, amelyet utasításbemenettel felül lehet bírálni.

Ha a **3. bit = 1**, akkor az FCB-ben levő állománynév-kiterjesztést csak akkor változtatja meg, ha az állományleírásban érvényes állománynév található.

Az elemzés befejeztét az állományleírás előtt, mögött vagy a leírásban levő, konvenció szerinti karakterek keresése követi. Ha a globális * karakterrel találkozunk, az a kidolgozottabb ? formátumra fordítódik.

Mint rendszerint, itt is az *AL* jelenti az eredményt: az $AL = 0$ jelzi, hogy sikeres volt egy állománnyal; az $AL = 1$ jelzi, hogy sikeres volt a globális karakterekkel (* vagy ?), amely figyelmeztet minket, hogy szükség van a "találd meg az elsőt, találd meg a következőt" feldolgozásra (☞ l. a 17-es és a 18-as függvényt); az $AL = 255$ (hexa FF) specifikációban lévő hibát jelent).

Az ismételt feldolgozások támogatására a *DS*:*SI* (valójában csak az *SI*) az elemzett bemenet utáni helyre mutat. Ha az elemzés sikertelen volt, akkor az FCB második bájtja ($ES:DI+1$) üres lesz.

A régi típusú általános DOS-szolgáltatáshoz hasonlóan, ez a függvény sem kezeli az útvonalneveket: ez ugyan korlátozza hasznosságát, de azért nem fölösleges.

A 42-es függvény (hexa 2A): dátumlekérdezés

A 42-es függvény jelzi a DOS aktuális dátumának értékét. A dátumot a *CX*-ben és a *DX*-ben kapjuk meg. A *DH* tartalmazza a hónap számát (1–12); a *DL* a napot (1–28, 29, 30 vagy 31 – ahogy ez megfelelő); a *CX* pedig az évet (1980–2099).

A hét napjait ez a szolgáltatás úgy jelzi, hogy az *AL*-regiszterbe 0-tól 6-ig terjedő értéket ad vissza, ezek jelentik a napokat vasárnaptól szombatig. A hét napjait a DOS mostoha gyerekként kezeli. A DOS-ban már az 1.10 változat óta jelen volt, de nincs megemlítve a 2.00 változatig. A 2.00 és a 2.10 kézikönyvben is hibás a leírása: az idő lekérdezési függvény részeként, nem pedig a dátumlekérdezési függvény részeként szerepel. Csak a DOS 3.00-tól kezdve írja le a gépkönyv annak, ami valójában. Ha használni akarjuk ezt a szolgáltatást, lapozzunk a fejezet végén levő példához!

A 43-as függvény (hexa 2B): dátumbeállítás

A 43-as függvény állítja be a DOS aktuális dátumát úgy, hogy a dátumát a 42-es függvényben jelezzük. A dátumot a *CX*-ben és a *DX*-ben állítjuk be. A *DH* tartalmazza a hónap számát (1 – 12), a *DL* a napét (1 – 28, 29, 30, vagy 31 – ahogy ez megfelelő); a *CX* az évet (1980 – 2099); az *AL* pedig a hét napját (0 – 6). (☛ További magyarázatát l. a 42. függvénynél. Ha használni akarjuk ezt a szolgáltatást, lapozzunk a fejezet végén lévő példához!)

A 44-es függvény (hexa 2C): időlekérdezés

A 44-es függvény jelzi az időpontot. Az időt a ROM-BIOS óraütem-számlálójából számolja ki (☛ l. a 12. fejezetben). A DOS reagál a ROM-BIOS „éjfél elmúlt" jelére, és 24 óránként frissíti a dátumot.

Az ütemszámláló értékét jelentéssel bíró időértékre konvertálja, és a *CX*-, ill. a *DX*-regiszterekben kapjuk vissza. A *CH* tartalmazza az órát (0 – 23, a 24 órás órán); a *CL* a perceket (0 – 59); a *DH* a másodperceket (0 – 59); a *DL* pedig a századmásodperceket (0 – 99).

Az óraütem tényleges sebessége kb. 0,054 másodperc, ezért a másodpercek századrészét nem lehet pontosan jelezni, és hozzávetőlegesen 1/20 másodpercnél nem is lehet pontosabban időzíteni. Ennek ellenére a másodpercek századrészének kiszámítására a DOS által használt algoritmus 100 értéket egyenletesen oszt el.

A DOS Technical Reference kézikönyvben megjelentetettekkel ellentétben, a hét napjait nem az időlekérdezés függvénye jelzi, hanem a 42-es (dátumlekérdezés) függvény, ahogy ezt józan ésszel elvárhatjuk.

A 45-ös függvény (hexa 2D): időbeállítás

A 45-ös függvény állítja be a napi időt. Az időt a *CX*- és *DX*-regiszterekben határozzuk meg. A *CH* tartalmazza az órát (0 – 23, a 24 órás órán); a *CL* a perceket (0 – 59); a *DH* a másodperceket (0 – 59); a *DL* pedig a századmásodperceket (0 – 99).

A 46-os függvény (hexa 2E): a lemezre írás ellenőrzésének beállítása

A 46-os függvény vezérli a lemezre írás műveleteinek ellenőrzését. Amikor az ellenőrzés be van kapcsolva, minden egyes lemezre írást egy ellenőrző olvasás követ. Ez nem hasonlítja össze az írott adatokat, hanem a megfelelő adatrögzítés CRC-jét (paritás-ellenőrzését) vizsgálja.

E szolgáltatás használatakor a *DL*-t 0-ra kell beállítani, ez azt sugallja, hogy ennek a függvénynek létezik néhány közzé nem tett változata is. Az *AL* 0-ra vagy 1-re állításakor kapcsolja ki vagy be az ellenőrzést.

A 2.00 vagy az annál magasabb DOS-változatokkal a 84-es (hexa 54) kibővített függvényt használhatjuk az aktuális ellenőrző-kapcsoló beállítások kijelzésére. (☛ L. a 17. fejezetet.)

Az állományvezérlő tömb (File Control Block, FCB)

A DOS hagyományos függvényhívásoknál a lemezegység állománykezelési feladatai a DOS állományvezérlő tömbje körül összpontosultak. Ez egy 44 bájtos terület, amely az általunk használt állományra vonatkozó leíró információkat tartalmazza. Számos függvényhívásból láthatjuk, hogy nemcsak hozzáférhetünk az FCB-hez, de csaknem teljesen vezérelhetjük is azt. Észrevehetjük azt is, hogy a 2.00 változattal bevezetett új DOS-függvények hívásai (amelyeket a 17. fejezet tárgyal), az állományra (vagy egyéb készülékre) vonatkozó vezérlő információk legnagyobb részét elrejti előlünk. Ehelyett csak egy egyszerű állományazonosító számmal dolgozunk – amelyet állomány foglaltsági jelnek (file handle) nevezünk –, a DOS minden egyéb feladatot elintéz. Az állomány foglaltsági jelet részletesen a következő fejezetben ismertetjük. Itt az FCB formátumát és használatát mutatjuk be.

Két fő része van az FCB-nek: maga az FCB, amely egy 37 bájtos terület; és a kiterjesztett FCB, amely az FCB-nek egy kvázi-opcionális, 7 bájtos előtagja. Az FCB legnagyobb része vezérlő információkat tárol az állományokról, beleértve az állománynevet, a lemez-

16.3. táblázat
Az állományvezérlő-tömb mezőinek leírása

Eltolás, dec	Méret, bájt	Beállítva	Leírás
−7	1	Us−1	„A kiterjesztés aktív"-jel: FF = igen, egyébként nem
−6	5	Us−1	Névlegesen nem használt: nullára kell állítani
−1	1	Us−1	Állományattribútum, amikor a kiterjesztés aktív
0	1	Us−1	Speciális lemezegységszám
1	8	Us−1	Állománynév vagy készüléknév
9	3	Us−1	Az állománynév kiterjesztés
12	2	Us−2	Aktuális tömbszám
14	2	Us−2	Rekordméret
16	4	DOS	Az állományméret bájtokban
20	2	DOS	Az állomány dátuma (bitkódolt, mint a tárgymutatóban)
22	10	DOS	A DOS egyéb vezérlő munkaterülete
32	1	Us−2	Az aktuális rekordszám („előjeles" bájt, l. a szöveget)
33	4	Us−2	Random rekordszám

egység-specifikációt, valamint a rekordméreteket és -számokat. A 7 bites kiegészítés jelzi, hogy az állománynak speciális állományattribútuma van. (☛ Az állományattribútumokról részletesebben az 5. fejezetben volt szó). Igen különös az FCB kiterjesztésekor kialakuló helyzet. A kiterjesztést csak akkor használjuk, amikor nem konvencionális attribútumú állományokkal dolgozunk, mint a rendszer- (vagy rejtett) állományok. Gondolhatjuk tehát, hogy a kiterjesztésre csak ilyenkor van szükség, és ha nem használunk speciális állományokat, nem kell tartalékolnunk a 7 bájtos tárterületet. Ez így nem igaz! A gyakorlatban kötelező az FCB kiterjesztésére területet félretenni, mert a kiterjesztés jelenléte az első bájtban a 255 (hexa FF) értékkel van jelezve. A helyzetet tovább súlyosbítva: ha nem használjuk az FCB kiterjesztését, meg kell győződnünk arról, hogy ezt jelöltük is. Ez azt jelenti, hogy a 7 bájtos terület használat közben közölje velünk: nincs rá szükség. Az ügyelet effajta tervezése zavarba ejti az ésszerűen gondolkodó programozókat, és mindenkit arra ösztönöz – beleértve a DOS tervezőit is –, hogy a 17. fejezetben tárgyalt kiegészítő függvények világosabb összeállításával próbálkozzanak.

Az FCB címzését a főrész kezdetétől számoljuk, és az FCB-n belül az összes mezőre, azoknak e mezőtől való eltolási értékével hivatkozunk. Ez az FCB kiterjesztést a −7 negatív eltolásra helyezi. Követni fogjuk a DOS konvencióit, a különböző mezőket e terminológia szerint írjuk le.

A 16.3. táblázatban bemutatott mezők közül azokat nem változtathatják meg programjaink, amelyeket a DOS állít be és vezérel. Az egyéb mezőket nekünk kell beállítanunk: amelyeket Us−1-gyel jelöltünk, az állomány megnyitása előtt kell beállítani, amelyeket pedig Us−2-vel, azokat az állomány olvasása ill. írása előtt. Az FCB részeinek felhasználását és kódolását olyan sorrendben tárgyaljuk, ahogy azok a táblázatban megjelennek.

A −7 és a −6 eltolás: ha használjuk az FCB kiterjesztést, a „kiterjesztés aktív" jelet 255-re (hexa FF) kell beállítani; ha pedig nem használjuk azt, akkor bármilyen más értékre. A DOS irodalma azt jelzi, hogy a következő 5 bájtos mezőt 0-ra kell beállítani, de lehet, hogy ez nem is lényeges.

A −1 eltolás: az állomány attribútumát a megnyitandó állomány speciális attribútumaira kell beállítani. Erre nincs szükség a szokásos állományoknál, amelyek attribútuma 0. A rejtett, a rendszer- és a katalógus-attribútumokat kell meghatározni. A címattribútum nem lényeges és nincs szükség attribútumspecifikációra egy csak olvasható állomány eléréséhez. (☛ Az attribútumkódolásról a további információkat l. az 5. fejezetben.)

A 0 eltolás: A 0 eltolásnál levő bájtban megadott speciális lemezegységszám jelzi, hogy melyik lemezegységgel kívánunk dolgozni. Ez nem ugyanaz, mint a szabványos lemezegységet azonosító szám, amelyben a 0 az *A*, az 1 pedig a *B* lemezegységet jelenti (és így tovább). Ehelyett ez a lemezegységszám egy rugalmasabb formátum, ahol az 1 az *A*, a 2 a *B* lemezegységet jelenti (és így tovább). Ez a rugalmasság abból származik, hogy a 0 használható az aktuális alaplemezegység jelzésére, bármi legyen is az. Mielőtt egy állományt megnyitunk, meghatározhatjuk azt a lemezegységet, amelyre szükségünk van, de jelölhetjük az alaplemezegységet 0-val is. Amikor az állományt megnyitottuk, a DOS a 0-t a specifikus lemezegységszámra változtatja, ezt aztán megfigyelhetjük a 25-es (hexa 19) függvény használatakor. Legyünk óvatosak, amikor ezzel a mezővel dolgozunk: ne keverjük össze a speciális lemezegységértékeket a konvencionálisabb lemezegységszámokkal, hiszen azok eggyel kisebb értékűek, mint az itt használt számok.

Az 1 és a 9 eltolás: az 1 és a 9 eltolásnál levő bájtok két mezője az állománynevet és a kiterjesztést adja. A konvenciót követve ezek a mezők balra rendezettek, jobb felől pedig üres (space) értékkel vannak feltöltve (**CHR$(32)**, hexa 20). A DOS konvencióit követve akár kisbetű, akár nagybetű használható. Ha az állománynév egy készüléknév, amelyet a DOS felismer (pl. CON:, AUX:, COM1:, COM2:, LPT1:, LPT2:, PRN: vagy NUL:), a DOS az illető készüléket használja a lemezállomány helyett.

🕮 MEGJEGYZÉS: Ezzel célszerűen rámutathatunk arra, hogy az FCB nem képes útvonalnevekkel dolgozni. Ha FCB-ket használunk, azok bármikor bármelyik lemezegységben az aktuális tárgymutatóra vonatkoznak. (☛ Az altárgymutatók és útvonalak rugalmasabb használatára l. az új kiterjesztett függvényeket a 17. fejezetben.)

A 12 és a 32 eltolás: Soros állományműveletekhez az aktuális tömböt és az aktuális rekordmezőket használjuk az állományon belüli hely nyomon követésére. E mezők használata meglehetősen furcsa. Ahelyett, hogy egy integrált rekordszámot használnánk, a rekordszám magas és alacsony értékű részre oszlik, amelyeket tömb- és rekordszámoknak nevezünk. Hogy a dolgok még csavarosabbak legyenek, a tömbök 128 rekordot tartalmaznak, az 1 bájtos rekordszámnál lehetséges 256 rekord helyett. Ezért előjeles a rekordszám a 16.3. táblázatban. Valójában ez nem is előjeles, mivel a negatív értékek nem megengedettek; csupán figyelmeztetés, hogy a megengedett értéktartomány 0-tól 127-ig, és nem 0-tól 255-ig terjed.

Az állomány első rekordja a 0. tömb 0. rekordja. Ezt a formátumot úgy tervezték, hogy lehetővé tegye minden egyes soros rekord helyének gyors kiszámítását. Ezzel a rekordelhelyezési rendszerrel a DOS lényegében a soros állományműveleteket a random állományműveletek egy változataként hajtja végre. Az első soros állományművelet előtt ezt a két mezőt vagy olvasásra, vagy írásra állítjuk be. A DOS minden egyes egymást követő művelettel eggyel megnöveli a mezőt. A mezők módosításával a soros állományban lépkedhetünk. Ez a taktika a bemeneti állományra remekül használható, a kimeneti állományra viszont rendkívül kockázatos.

A 14 eltolás: A 14 eltolásnál kezdődő rekordméretmező az állomány logikai rekordjainak méretét adja meg bájtokban. Amikor programjaink arra kérik a DOS-t, hogy egy rekordot írjon vagy olvasson, a logikai rekordméret a DOS lemezterülete és a programunk adatterülete között átvitt bájtok száma. Ennek az értéknek semmi köze az állományhoz abban az értelemben, ahogy azt a DOS látja, vagy ahogy az állományt tároljuk a lemezen. Csupán azt mutatja meg, hogyan kívánja ebben az időpontban ez a program használni az állományt. Ugyanazon az állományadaton különböző változatú rekordméretekkel is dolgozhatunk. Ha nem építettük az állományt, akkor rendszerint úgy kezeljük, mintha a rekord 1 bájt hosszúságú lenne; azért, hogy helyet takarítsunk meg. Néhány általános szövegszerkesztő program szokásosan az ASCII-állományoknál 128 bájt méretű rekordokat használ. Ez kevesebb, mint századrészére csökkenti azoknak a DOS függvényhívásoknak a számát, amelyek az állomány kezeléséhez szükségesek. Ezért a trükkért természetesen némi árat is kell fizetnünk: a DOS által rögzített állományméret akár 127 bájttal is nagyobb lehet; és e programoknak egy kis extra alapozó munkát kell végezniük ahhoz, hogy az adatokat rendben tartsák. Amikor a 15-ös függvény segítségével egy állományt megnyitunk, a DOS a rekordméretet a 128 bájtos alapméretre állítja. Ha más méretet akarunk – pl. 1 bájtos műveleteknél – az állomány megnyitása után ezt meg kell változtatnunk.

A 16 eltolás: A 16 eltolásnál levő állományméret-mező jelzi az állomány méretét bájtokban. Az értéket a tárgymutatóból az FCB-be másoljuk, amikor a DOS megnyit egy állományt. Kimeneti állománynál a DOS e mező értékét az állomány növekedésével összhangban változtatja. Az állomány lezárásakor az értéket a tárgymutató megfelelő helyére másolja. E mező változtatása lehetővé teszi, hogy az utolsó pillanatban a kimeneti állomány méretén némi vezérlést kapjunk, de ezt körültekintően tegyük. Ha egy katalógust állományként kíséreljük meg beolvasni, ez a mező 0 lesz, amikor az állomány megnyílik. (De ne akarjuk az FCB kiterjesztést használni arra, hogy katalógus állományattribútumát jelezzük, mivel a katalógusbevitel 0 hosszúságot mutat.) Ha az alkatalógust sikeresen akarjuk kiolvasni, ezt a mezőt valamilyen önkényesen nagy értékre kell beállítanunk. Még valamire gondolnunk kell, amikor

az átnevezési műveletet (23-as függvény) használjuk: az új állománynevet 16 eltolással helyezzük el, pontosan oda, ahol az állományméret-mező helyezkedik el.

A 20 eltolás: az állomány dátumát 2 bájtos mezőben kódoljuk ugyanolyan formában, mint a katalógus belépési pontokat; az MM/DD/YY (hónap/nap/év) formátumot használjuk. Ezt a mezőt a DOS azzal az információval állítja be, amelyet az állomány megnyitásakor a katalógusból vesz. (☛ A dátum kódolásáról a további információkat l. az 5. fejezetben.)

A 33 eltolás: a random rekordmező a random olvasás és írás műveletek alatt használatos éppúgy, mint ahogy az aktuális rekord- és tömbszámok a soros műveleteknél. Ennek a mezőnek a formátuma egy 4 bájtos, 32 bites egész szám (integer), amelyet könnyen fel lehet bontani bájtokra vagy szavakra. A rekordok 0-tól számozottak, ezért könnyen kiszámíthatjuk az állományeltolást bármely rekordra, ha a random rekordszámot a rekordmérettel megszorozzuk. Ezt a mezőt randomállomány-művelet előtt be kell állítanunk. A DOS a mezőt változatlanul hagyja.

Egy példaprogram

Ennek a fejezetnek assembly nyelvű példája érdekes és ravasz feladat. Ezt a rutint a szerző (Norton Utility) programjaiban való használatra fejlesztette ki, így egy ténylegesen működő kódot láthatunk, amelyen csak a megjegyzések miatt módosítottunk.

A rutin kiszámolja, hogy a hét melyik napja lesz a DOS munkatartományának bármelyik napja; a tartomány 1980. január 1-én, kedden indul, és 2099. december 31-én, csütörtökön fejeződik be. Eseteként hasznos lehet, ha a program képes jelezni a hét napját az aktuális dátumra vagy bármely más dátumra vonatkozóan. A DOS pl. nyomon követi minden egyes állomány dátumát és időpontját az utolsó változtatás óta. Mivel gyakran használjuk ezt az információt arra, hogy megtudjuk, mikor dolgoztunk legutoljára az állománnyal, hasznos lehet tudni a hét napját is. Valójában a hét napja gyakran több jelentéssel bír, mint a tényleges dátum.

Bár számos érdekes és okos algoritmust publikáltak a hét napjának kiszámítására, e programot megírni valójában elég unalmas. Úgy tűnik, hogy amikor szükség lenne rájuk, a megjelent algoritmusok ritkán vannak kéznél; ha viszont kéznél vannak, gyakran programjaink számára nem alkalmas nyelven írták őket. A 2.00 változattól kezdődően a DOS egy hét napja számítást is magába foglal, ez megkímél minket attól a fáradságtól, hogy sajátunkat megírjuk. A DOS-program számunkra csak olyan formában érhető el, amely a hét aktuális napját jelzi, ez azonban nem akadály: időlegesen megváltoztathatjuk a DOS dátumát a minket érdeklő dátumra, és ekkor a DOS jelzi a hét napját. Ezt végzi el a következő assembly nyelvű rutin.

Ez a rutin azért is érdekes, mert három együttműködő DOS-függvényhívást mutat be. Bemutatja azokat az apró finomságokat, amelyek ezek veremben való tárolásához és visszanyeréséhez kapcsolódnak. Látni fogjuk, hogy a veremhasználatot minden esetben gondosan meg kell szervezni, nehogy a különböző értékek egymás útjába kerüljenek.

Ezt a WEEKDAY-nek (a hét napjának) nevezett speciális szubrutint a Lattice/Microsoft C fordítóval való használat igényei szerint készítették el. A rutint három integer változóval hívjuk, ezek adják meg a minket érdelő hónapot, napot és évet. A rutin a hét napját integer formájában adja vissza, a 0-tól 6-ig terjedő tartományban (ez vasárnaptól szombatig jelzi a napokat). Ez jól megfelel a C nyelv mátrixokra vonatkozó konvenciójának, indexet adva a stringek egy tömbjéhez, amely a napok nevét tartalmazza. Ezt a szubrutint ezért a következőképpen használhatnánk:

DAY_NAMES (WEEKDAY (MONTH, DAY, YEAR))

Tudnunk kell azonban, hogy ez a rutin a dátumot vakon kezelve dolgozik, nem vizsgálja sem a dátum érvényességét, sem azt, hogy a dátum tartományát a DOS elfogadja-e. Meg kell jegyeznünk, hogy ez a rutin a 2.00 vagy a magasabb DOS-változatokat igényli. Az alábbiakban következik a szubrutinunk:

```
          PGROUP    GROUP PROG
          PUBLIC    WEEKDAY
PROG      SEGMENT   BYTE PUBLIC 'PROG'
          ASSUME    CS:WEEKDAYS
WEEKDAY   PROC      NEAR
; Stratégia: régi dátum lekérdezése és elmentése
          ; az új dátum beállítása
          ; annak a dátumnak a lekérdezése, amely a hét napját jelzi
          ; a régi dátum visszaállítása
PUSH      BP
MOV       BP,SP
MOV       AH,2AH      ; a dátum lekérdezése (kimentése)
INT       21H         ; DOS-függvény hívás
PUSH      CX          ; a régi dátum kimentése...
PUSH      DX          ; ... a verembe
MOV       CX, [BP+8]  ; az év
MOV       DL, [BP+6]  ; a nap
MOV       DH, [BP+4]  ; a hónap
MOV       AH,2BH      ; a dátum időleges beállítása
INT       21H         ; a DOS-függvény hívás
MOV       AH,2AH      ; a dátum lekérdezése, a hét napjának megállapításához
INT       21H         ; DOS-függvény hívás
POP       DX          ; a régi dátum visszaállítása
POP       CX          ; ...a veremből
PUSH      AX          ; a hét napjának kimentése a verembe
MOV       AH,2BH      ; a régi dátum visszaállítása
INT       21H         ; DOS-függvény hívás
POP       AX          ; a hét napjának visszahozása a veremből
MOV       AH,0        ; a felső rész törlése, hogy integer legyen
POP       BP
RET
WEEKDAY   ENDP
PROG      ENDS
END
```

17 A DOS-függvények

Új DOS-függvények

Bevezetés • 267

DOS 2 bővítések • 267

A kiterjesztett DOS-függvények bővítései • 267

Installálható készülékmeghajtók • 270

A kiegészített DOS-függvények • 270

A 47-es függvény (hexa 2F): a DTA-cím lekérdezése • 272
A 48-as függvény (hexa 30): a DOS-változatszám lekérdezése • 272
A.49-es függvény (hexa 31): **KEEP**: a „program befejezése memóriában maradással" függvény továbbfejlesztése • 273
Az 51-es függvény (hexa 33): a *Ctrl-Break* lekérdezése vagy beállítása • 273
Az 53-as függvény (hexa 35): a megszakításvektor lekérdezése • 273
Az 54-es függvény (hexa 37): a szabad lemezterület lekérdezése • 274
Az 56-os függvény (hexa 38): az országfüggő információk lekérdezése • 274
Az 57-es függvény (hexa 39): **MKDIR**: katalógus létrehozása • 277
Az 58-as függvény (hexa 3A): **RMDIR**: katalógus törlése • 277
Az 59-es függvény (hexa 3B): **CHDIR**: aktuális katalógus megváltoztatása • 277
A 60-as függvény (hexa 3C): **CREAT**: állomány létrehozása • 277
A 61-es függvény (hexa 3D): állomány megnyitása • 278
A 62-es függvény (hexa 3E): az állományfoglaltság lezárása • 279
A 63-as függvény (hexa 3F): olvasás állományból vagy készülékről • 279
A 64-es függvény (hexa 40): írás állományba vagy készülékre • 280
A 65-ös függvény (hexa 41): állomány törlése • 280
A 66-os függvény (hexa 42): az állománymutató mozgatása • 280
A 67-es függvény (hexa 43): **CHMOD**: állományattribútumok lekérdezése vagy beállítása • 281
A 68-as függvény (hexa 44): **IOCTL**: I/0 berendezések vezérlése • 281
A 69-es függvény (hexa 45): **DUP**: állományfoglaltság másolása • 283

A 70-es függvény (hexa 46): **CDUP**: a foglaltság másolásának kikényszerítése • 283
A 71-es függvény (hexa 47): az aktuális katalógus lekérdezése • 284
A 72-es függvény (hexa 48): memória lefoglalása • 284
A 73-as függvény (hexa 49): lefoglalt memória felszabadítása • 284
A 74-es függvény (hexa 4A): **SETBLOCK**: a lefoglalt memóriatömb méretének módosítása • 284
A 75-ös függvény (hexa 4B): **EXEC**: program betöltése és végrehajtása • 285
A 76-os függvény (hexa 4C): a folyamat befejezése • 286
A 77-es függvény (hexa 4D): az alprogram visszatérési kódjának lekérdezése • 287
A 78-as függvény (hexa 4E): **FIND FIRST**: az állománykeresés indítása • 287
A 79-es függvény (hexa 4F): **FIND NEXT**: az állománykeresés folytatása • 288
A 84-es függvény (hexa 54): az ellenőrző állapot lekérdezése • 288
A 86-os függvény (hexa 56): állományátnevezés • 288
A 87-es függvény (hexa 57): az állomány dátumának és időpontjának lekérdezése vagy beállítása • 289

DOS 3.00 bővítések • 289

A 89-es függvény (hexa 59): kiterjesztett hibakód lekérdezése • 289
A 90-es függvény (hexa 5A): időleges állomány létrehozása • 292
A 91-es függvény (hexa 5B): új állomány létrehozása • 292
A 92-es függvény (hexa 5C): állományhozzáférés lezárása vagy megnyitása • 292
A 98-as függvény (hexa 62): a PSP-cím lekérdezése • 293

Bevezetés

Az előző fejezetben a DOS-szolgáltatásokra összpontosítottunk, ebben a fejezetben viszont azokat a DOS-függvényeket fogjuk tárgyalni, amelyek még a DOS fejlettebb változatainál, a 2.00-nál és az azt követőknél újdonságok. A 16. fejezetben tárgyalt régi (vagy univerzális) szolgáltatásokat a DOS terminológia szerint hagyományos függvényeknek, az e fejezetben tárgyalt új szolgáltatásokat pedig kiterjesztett fügvényeknek nevezzük.
A DOS 2.10 kiterjesztett függvényei azonosak a DOS 2.00-éval. A DOS 3.00 változata bevezetett néhány változtatást a már meglevő függvényekre, és hat új függvénnyel is bővült. Ebben a fejezetben minden DOS 2 függvényt részletesen leírtunk, ill. ahol a DOS 3.00 bővítések megjelennek, ott azokat is. Az új DOS 3.00 kiterjesztett függvényeket részletesen e fejezet végén ismertetjük. A 15. fejezetben találhatók azok a megszakítások, amelyeket nem függvényként tartunk számon. A 16. fejezetben azokról a hagyományos függvényhívásokról van szó, amelyeket bármely DOS-változattal használni lehet.

DOS 2 bővítések

Csaknem minden új DOS-változatnál megváltozik a DOS működési módja és a programozóknak biztosított szolgáltatások száma is. A DOS 2.00 bevezetése hozta létre a legdrámaibb változtatásokat. A meglevő 42 szolgáltatáshoz további 33 új szolgáltatást adott. Az új szolgáltatások eredményeként megváltoztatta az állományinformációhoz való hozzáférésünket, és az installálható készülékmeghajtóknak nevezett programok használatával lehetővé tette, hogy a DOS csaknem minden hardverkészülékkel együtt dolgozhasson. Mielőtt részletesebben tárgyalnánk a kiterjesztett függvényeket, röviden kitérünk arra, hogyan befolyásolják programozási gyakorlatunkat ezek a bővítések.

A kiterjesztett DOS-függvények bővítései

A legtöbb DOS 2.00-val és DOS 3.00-val bevezetett kiterjesztett szolgáltatásnak három olyan új jellemzője van, amely közvetlenül hat e szolgáltatások felhasználási módjára.

Először: ezeknek a függvényeknek a legtöbbje az *AL*-regiszterbe egy szabvány hibakódot ad vissza. *Másodszor:* mindazok a függvények, amelyek stringbemenetet használnak, speciális stringformátumot kívánnak, amelyet ASCIIZ-formátumnak nevezünk. (Ez egy olyan string, amelyet egy nullákból álló bájt követ.) *Harmadszor:* e DOS-függvények közül a legtöbb az FCB helyett egy 16 bites számot használ – amelyet állomány foglaltsági jelzőnek (file handle) nevezünk –, hogy nyomon kövesse azokat az állományokat és I/0 berendezéseket, amelyekkel a program adatot cserél. A következő néhány oldalon e bővítéseket egymás után részletesen tárgyaljuk.

Szabványos hibakódok

A kiterjesztett DOS-függvények közül a legtöbb, ha befejezte működését, egy szabványos bináris hibakódot ad vissza, amelyet visszaadott kódnak nevezünk. Ezeket a 17.1. táblázat sorolja fel. Általában a visszaadott kódok jelzik azt, hogy a szolgáltatás valamilyen okból

17.1. táblázat
A sikertelen függvényművelet után az AX-regiszterben visszaadott szabványos hibakódok és a hozzájuk rendelt jelentések

Hibakód		Jelentés
Dec	Hexa	
1	1	Érvénytelen függvényszám
2	2	Az állományt nem találja
3	3	Az utat nem találja
4	4	Nincs szabad állományfoglaltsági jelző, mind használatban van
5	5	A hozzáférés megtagadva
6	6	Érvénytelen állományfoglaltság
7	7	A memóriavezérlő tömbök tönkrementek
8	8	Nem elegendő a memória
9	9	Érvénytelen memóriatömb címe
10	A	Érvénytelen környezet (l. a DOS SET-utasítását)
11	B	Érvénytelen formátum
12	C	Érvénytelen hozzáférési kód
13	D	Érvénytelen adat
14	E	Nem használt
15	F	Érvénytelen a lemezegység megadása
16	10	Kísérlet az aktuális tárgymutató elmozdítására
17	11	Nem ugyanaz a készülék
18	12	Nincs több állomány

meghiúsult. Programjaink így meghatározhatják a probléma jellegét, és a követendő legmegfelelőbb tevékenységet.

A szabványos visszaadott kódok az *AX*-regiszterben kerülnek vissza (tulajdonképpen a fél regiszter is elég lenne, mivel az összes kód értéke jóval 255 alatt van). A *CF* (carry flag) használatos a hiba jelzésére – legalábbis elméletben. Ezekkel a szolgáltatásokkal kísérletezve felfedezhetjük, hogy nem függhet a *CF*-től azoknak a feltételeknek a jelzése, amelyekről úgy gondoljuk, hogy hibajelet hív. Szerencsére a visszaadott kódok megbízhatóak, és bármikor számíthatunk a 17.1. táblázatban bemutatott értékek használatára, ha egy függvény hibát ad vissza. Nem dönthető el ilyen egyszerűen, hogy a *CF* jelzőbitet jelzésként használhatjuk-e vagy sem. Mindenképpen vizsgáljuk meg kísérleti úton a függvényt, hogy a *CF*-et hogyan változtatja vagy hagyjuk teljesen figyelmen kívül, csak azokat a kódokat vizsgálva, amelyek az *AX*-regiszterben megjelenhetnek. Azoknak a függvényeknek a leírásában, amelyek megadják a visszaadott kódokat, megtaláljuk azoknak a kódoknak a rövid leírását is, amelyeket a legvalószínűbb, hogy egy függvényhiba visszaad.

🕮 MEGJEGYZÉS: A DOS 3.00 nemcsak a szabványos hibakódokat nyújtja, de a 89-es függvény útján átfogóbb hibakódokat is.

Az ASCIIZ-string

Ha bármelyik kiterjesztett függvény váltakozó hosszúságú stringet igényel (pl. az útvonalnevet), az ASCIIZ-nek nevezett stringformátumot használjuk. Az ASCIIZ-string konvencionális ASCII-karakterek sorozatából ál, amelyet egy zéró bájt követ, jelezve a string végét. Egy jellemző útvonalnév–string pl. a következőhöz hasonló formát veheti fel:

A: \DIRECTORY1 \DIRECTORY2 \FILENAME.EXT(NULLA BYTE)

A lemezegység specifikációja opcionális, szeparátorként akár előre (/), akár visszafelé dőlő (\) perjel elfogadott.

Az ASCIIZ-string formátumát az UNIX operációs rendszer és a C nyelv is általánosan használja; ez csak egyike azoknak az új C (UNIX) stílusú elemeknek, amelyet a DOS 2 változatokkal vezettek be.

Az állományfoglaltsági jelzők

A DOS kiterjesztett függvényei az állományok kezelését sokkal inkább „belügynek" tekintik, mint a hagyományos DOS-szolgáltatások. A hagyományos szolgáltatások az állományvezérlő tömböt (FCB) használják, ehhez van hozzáférésünk, és felette csaknem teljes a felügyeletünk. A kiegészített szolgáltatások az állományra (vagy berendezésre) vonatkozó vezérlő információk nagy részét „szemérmesen", előlünk elrejtve kezelik.

A hagyományos szolgáltatásoknál mi jelöljük ki a DOS egy FCB-re mutatásával, hogy melyik állománnyal dolgozunk. Ugyanezeket a szolgáltatásokat használjuk a szokásos állomány-háztartási feladatokra is, pl. az állomány megkeresésére, az állomány méretének meghatározására. Ezzel ellentétben a kiterjesztett szolgáltatásoknál állományfogókkal dolgozunk, ez nem egyéb, mint egy egyszerű 16 bites szám. A DOS ezzel a számmal azonosítja azt az állományt vagy berendezést, amellyel dolgozunk, és a szokásos állomány-karbantartási műveletek legtöbbjét is elvégzi.

17.2. táblázat
Az öt szabványos foglaltsági kód

Foglaltság kódja	Használat	Alapérték
0	Szabványbemenet (általában billentyűzetbemenet)	CON:
1	Szabványkimenet (általában monitorkimenet)	COB:
2	Szabványos hibakimenet (minden esetben a képernyő)	CON:
3	Szabvány kiegészítő berendezés (AUX: berendezés)	AUX:
4	Szabvány nyomtató (LPT1: vagy PRN: berendezés)	PRN:

A DOS fenntartja a foglaltsági jelzők feletti teljes vezérlést, ő adja ki a jelzőszámokat, valahányszor állományt hozunk létre vagy nyitunk meg. Öt szabványos jelző van, ezeket 0-tól 4-ig számozzuk, és minden programhoz automatikusan rendelkezésünkre állnak (l. a 17.2. táblázatot). A nagyobb számokat a DOS adja ki igény szerint.

A hagyományos függvények 99 állomány (és állomány–FCB) egyidejű nyitását teszik lehetővé a DOS **FILES**-utasításának használatával. (AT Az AT-nál 255 a megengedett.) A DOS 2 változat alatt működő kiegészített függvények mégis legfeljebb 20 foglaltsági jelzőt adnak ki bármely programnak, ezáltal korlátozva azoknak az állományoknak a számát, amelyeket a program ténylegesen meg tud nyitni. Igény szerint a DOS 3.00 többet is adhat, a DOS 2.00 alapértelmezése a 20-as korlát. Ez a korlátozás a kiterjesztett függvényekben valószínűleg csak az igen nagy bonyolultságú programokra hat, mert azok egyidőben nagyszámú nyitott állományt igényelnek.

Installálható készülékmeghajtók

A DOS 2.00-val bevezetett két pótlólagos jellemzőre is érdemes odafigyelnünk. Ezek az installálható készülékmeghajtók, és ezeken belül az ANSI-meghajtó (amelyet rendszerint ANSI-SYS-nek nevezünk). Ez a készülékmeghajtó számos DOS-adaptációhoz szabvány szerint jár.

A DOS készülékmeghajtói olyan programok, amelyek anélkül teremtenek kapcsolatot a hardverkészülékek és a számítógépünk között, hogy újraírnánk a BIOS-t. A készülékmeghajtó lehet egy rutin, amely új hardver hozzáadását támogatja, pl. a botkormányét (joystick) vagy az egészét (mouse); de lehet olyan rutin is, amely úgy módosítja a szabványos hardver működését, hogy az olyan feladatokat is végre tudjon hajtani, amelyeket a DOS-szolgáltatások nem. (Az ANSI-meghajtó az utóbbi típusnak egy példája; megengedi, hogy a képernyőkimenetet vagy a billentyűzetbemenetet módosítsuk a BIOS-szolgáltatás rutinjainak használata nélkül.) Mivel a készülékmeghajtókat a DOS részeként alkották, programunknak nem kell lenyúlnia a BIOS-hoz vagy a hardverszintre, hogy új készülékeket helyezzen el. Ez fontos jellemző az olyan programok számára, amelyek ablak vagy multitasking környezetben működnek.)

A készülékmeghajtók nem kapcsolódnak közvetlenül a kiterjesztett DOS-függvényekhez, ezért részletesebben az *A* függelékben tárgyaljuk őket. A készülékmeghajtókat ugyan csak a könyv végén ismertetjük, ez azonban jelentőségüket nem csökkenti. Minden programozó, aki a programjai hosszúsága miatt aggódik, ismerje meg a készülékmeghajtók használatát és működését. (☛L. az *A* függeléket.)

A kiegészített DOS-függvények

Az összes kiegészített DOS függvényhívás a 33-as (hexa 21) megszakítás révén érhető el. Az egyes függvényeket a függvényszám *AH*-regiszterbe helyezésével választjuk ki. A kiegészített függvényeket használó bármelyik programnak először a DOS-változat számát kell megvizsgálnia, hogy meggyőződjék: a függvények a rendelkezésére állnak. A 48-as (hexa 30) kibővített függvény nyújtja ezt a szolgáltatást. A függvények a 17.3. táblázatban mutatott logikai csoportba szerveződnek. Attól a gondolattól vezérelve, hogy a függvényhívások logikai csoportosítását a lehető legvilágosabbá tegyük, a DOS Technical Reference kézikönyvtől kissé eltérve rendszereztük és írtuk le azokat. Ha össze akarjuk hasonlítani ezt a rendszerezést az IBM-ével, győződjünk meg, hogy nem értjük-e félre. A 17.4. táblázat sorolja fel az egyedi függvényhívásokat.

17.3. táblázat

A kiegészített DOS-függvényhívások logikai csoportjai

Függvény		Csoport
Dec	Hexa	
47–56	2F–38	Kibővített függvények csoportja
57–59	39–3B	Tárgymutató-csoport
60–70	3C–46	Kibővített állomány-karbantartási csoport
71	47	Tárgymutató-csoport
72–75	48–4B	Kibővített memória-karbantartási csoport
76–79	4C–4F	Kibővített függvények csoportja
84–87	54–57	Kibővített függvények csoportja
88–98	58–62	DOS 3.00 kiegészítések

17.4. táblázat

A 33-as megszakítással meghívott és az AH-regiszterben kiválasztott kiegészített DOS-függvényhívások

Függvény		Leírás
Dec	Hexa	
47	2F	A DTA-cím lekérdezése
48	30	A DOS-változatszám lekérdezése
49	31	**KEEP**: a „program befejezése memóriában maradással" függvény továbbfejlesztése
51	33	A *Ctrl-Break* lekérdezése vagy beállítása
53	35	A megszakításvektor lekérdezése
54	36	A szabad lemezterület lekérdezése
56	38	Az országfüggő információk lekérdezése
57	39	**MKDIR**: katalógus létrehozása
58	3A	**RMDIR**: katalógus törlése
59	3B	**CHDIR**: aktuális katalógus megváltoztatása
60	3C	**CREAT**: állománylétrehozás
61	3D	Állománymegnyitás
62	3E	Az állományfoglaltság lezárása
63	3F	Olvasás állományból vagy készülékről
64	40	Írás állományba vagy készülékre
65	41	Állománytörlés
66	42	Az állománymutató mozgatása
67	43	**CHMOD**: állományattribútumok lekérdezése vagy beállítása
68	44	**IOCTL**: I/O berendezések vezérlése
69	45	**DUP**: állományfoglaltság másolása
70	46	**CDUP**: a foglaltság másolásának kikényszerítése
71	47	Az aktuális katalógus lekérdezése

Függvény		Leírás
Dec	Hexa	
72	48	Memória lefoglalása (allokálás)
73	49	A lefoglalt memória felszabadítása
74	4A	SETBLOCK: a lefoglalt memóriatömb méretének módosítása
75	4B	Program betöltése és végrehajtása
76	4C	A folyamat befejezése
77	4D	Az alprogram visszatérítési kódjának lekérdezése
78	4E	FIND FIRST: az állománykeresés indítása
79	4F	FIND NEXT: az állománykeresés folytatása
84	54	Az ellenőrző állapot lekérdezése
86	56	Állományátnevezés
87	57	Az állomány dátumának és időpontjának lekérdezése vagy beállítása
89	59	A kiterjesztett hibakód lekérdezése
90	5A	Időleges állomány létrehozása
91	5B	Új állomány létrehozása
92	5C	Állományhozzáférés lezárása vagy megnyitása
98	62	A PSP-cím lekérdezése

A 47-es függvény (hexa 2F): a DTA-cím lekérdezése

A 47-es függvény adja vissza azt a lemezátviteli területet (vagy DTA-címet), amelyet a DOS éppen használ. A cím az *ES*:*BX* regiszterpárba kerül. Mivel az *ES*-regisztert a *DS*-regiszter helyett a szegmensrész visszaadására használjuk, ez a szolgáltatás nem ütközik a program normál működésével. Állítsuk ezt szembe a 16. fejezetben tárgyalt 26 függvénnyel!

A 48-as függvény (hexa 30): a DOS-változatszám lekérdezése

A 48-as függvény adja vissza a DOS kis és nagy változatszámait. A nagy változatszámok az *AL*-be (a DOS 2.10 esetén ez a 2-es), a kis változatszámok pedig az *AH*-ba (10-et a DOS 2.10 esetén) kerülnek. Ez a DOS fejlettebb változatainál új szolgáltatás, még nem létezik a DOS 1 változatainál. Szerencsére, ha a 48-as függvényt használjuk, bármely DOS 1 változattal következetesen azonosítható eredmény jelenik meg. Fő változatszámként az *AL*-regiszterben 0 található. Sajnos, a kis változatszám nem megbízható a DOS 1 változatoknál.

A fejlettebb jellemzőket használó bármelyik program a megfelelő DOS-változat használatát képes egy függvényhívással ellenőrizni. Ellenőrzéskor megkülönböztetett figyelemmel vizsgálja meg a nagy változatszámot, hogy az nagyobb vagy egyenlő-e 2-val (vagy 3-mal), ha a DOS 3 szolgáltatás használja. Ha a függvény egy 2-nél (vagy 3-nál) kisebb számot jelez vissza, a programnak udvariasan be kell fejeződnie.

A 49-es függvény (hexa 31): KEEP: a „program befejezése memóriában maradással" függvény továbbfejlesztése

A 49-es függvény az általános 39-es megszakítás, a „program befejezése memóriában maradással" szolgáltatás fejlettebb változata. A program befejezésén kívül a 49-es függvény lehetővé teszi a befejeződő programnak azt is, hogy egy visszaadott kódot jelezzen, amely az *AL*-regiszterbe kerül, és a kötegelt feldolgozás **ERROR LEVEL**-utasításával vizsgálható.

Az ilyen módon véget érő programoknak közölniük kell a DOS-szal, hogy belőlük menyi maradjon a memóriában, és mennyit lehet eldobni. Ez a szegmensparagrafus érték *DX*-be töltésével megy végbe, amely meghatározza a rezidensrész végét közvetlenül követő memóriabekezdést. Ez vezérli, hogy a program eredeti memóriaterületéből mennyi tartandó meg és mennyi szabadítandó fel.

A program további memóriát igényelhet a memóriaelhelyezési szolgáltatások (a 72–75. függvények) használatával. Az e szolgáltatásokkal kiutalt memóriákat nem szabadítja fel a *DX*-ben meghatározott érték. A *DX* csak azokat a memóriákat követi nyomon, amelyeket a DOS akkor jelöl ki, amikor a programot először betöltjük. Ez a kiosztás nem áll a program vezérlése alatt. Ha szabadon osztunk ki memóriákat a rezidens programban, az olyan, mintha vékony jégen korcsolyáznánk, de óvatosan ez is kivitelezhető.

Az 51-es függvény (hexa 33): a *Ctrl-Break* lekérdezése vagy beállítása

Az 51-es függvény vagy jelenti, vagy vezérli a *Ctrl-Break* feldolgozási állapotát. Emlékezzünk vissza, hogy a DOS a *Ctrl-Break*re csak nagyon korlátozottan hat. (☛ L. a 15. fejezetet.) A DOS 2.00 változattól kezdve az egyik legtökéletesebb megoldás a *Ctrl-Break*-vizsgálat kiterjesztése úgy, hogy az bármely DOS-művelet altt megtörténjék. Az egyes DOS-változatokkal való kompatibilitás érdekében ennek a kiterjesztett *Ctrl-Break*-vizsgálatnak opcionálisnak kel lennie. Ezt három módon lehet vezérelni: a **BREAK**-utasítással, a CONFIG.SYS inicializáció állománnyal és a DOS-függvénnyel. Ezek teszik lehetővé bármelyik programnak, hogy elvégezzék a *Ctrl-Break* vizsgálatát.

Ha az *AL*-et 0-ra állítjuk, akor lekérdezhetjük az aktuális megszakítási állapotot, amelyet a *DL*-ben kapunk vissza. A $DL=0$ azt jelenti, hogy a megszakításbillentyű ellenőrzése ki van kapcsolva; a $DL=1$ pedig azt, hogy be van kapcsolva. Ha az *AL*-et 1-re állítjuk, akkor beállíthatjuk a megszakításállapotokat e kódok használatával a *DL*-ben is. A $DL=0$ hatályon kívül helyezi a válaszadást megszakításbillentyűre, azaz a *Ctrl-Break* nem jut érvényre; a $DL=1$ pedig lehetővé teszi a megszakításbillentyű kezelését.

Az 53-as függvény (hexa 35): a megszakításvektor lekérdezése

Az 53-as függvény az *AL*-regiszterben levő megszakításszámra vonatkozó megszakításvektort adja vissza. A vektort az *ES:BX* regiszterpárban kapjuk vissza.

E szolgáltatásnak számos felhasználása van. A legkézenfekvőbb: az aktuális megszakításvektornak a 37-es (hexa 25) függvénnyel való megváltoztatása előtti megőrzése úgy, hogy

később azt vissza lehessen állítani. Egy másik felhasználás: egyszerűen csak az aktuális vektort ellenőrzi; pl. arra is használhatjuk ezt a függvényt, hogy megnézzük változott-e az alapvektor beállítása. Különösen hasznos felhasználás a megszakításvektor olyan beállításainak megismerése, amelyek megszakításkezelő szubrutinok helyett táblákra mutatnak. A vektorbeállítással egyszerűbb a táblainformációhoz hozzáférni.

Az 54-es függvény (hexa 37): a szabad lemezterület lekérdezése

Az 54-es függvény nemcsak a szabad területről (amelyet a szolgáltatás neve amúgy is sugall), hanem a lemezterület állapotáról is rengeteg érdekes és hasznos információt nyújt. Mielőtt e szolgáltatást hívnánk, a *DL*-regiszterrel kiválasztjuk a minket érdeklő lemezegységet: a $DL=0$ jelzi az alap-lemezegységet; a $DL=1$ az *A* lemezegységet; a $DL=2$ a *B* lemezegységet stb. Jegyezzük meg a különbséget e jelzés és a konvencionálisabb $0=A$ lemezegységjelzés között. Az öszes DOS-szolgáltatáshoz hasonlóan, de eltérően a ROM–BIOS-szolgáltatásoktól, ezek logikai, nem pedig fizikai lemezegységek.

Ha valamilyen hiba van – pl. egy érvénytelen lemezegység –, azt az *AX*-regiszterben hexa FFFF-el jelzi. Egyébként az *AX* tartalmazza, hogy egy kiutalásban (cluster) hány szektor van, a *CX* a szektoronkénti bájtokat, a *BX* a felhasználható klaszterek számát, a *DX* pedig az összes klaszter számát. Ezekből a számokból sok érdekeset megtudhatunk a lemezről.

Ezeket a képleteket pl. az alábbiak kiszámítására használhatjuk:

```
CX * AX            'a klaszterenkénti bájtok száma
CX * AX * BX       'a szabad bájtok teljes száma
CX * AX * DX       'a teljes tárterület bájtokban
(BX * 100)/DX      'a szabad terület százaléka
```

Ha az *S* lenne egy állomány mérete bájtokban, az elfoglalt szektorok számát a következőképpen tudnánk kiszámolni:

```
(S + CX * AX - 1) / (CX * AX)
```

Hasonló képlet adná meg számunkra a kiutalt egységek számát, valamint az állományhoz kiutalt, de nem használt terület összegét és arányát.

Az 56-os függvény (hexa 38): az országfüggő információk lekérdezése

Érdekes szolgáltatást nyújt: az 56 függvény lehetővé teszi a programnak, hogy automatikusan beállítsa a különböző nemzetközi pénzegység és időformátum konvenciókat, a DOS által adott országspecifikus információk táblázatának használatával. E szolgáltatás DOS 2 változata az országinformációknak csak kis készletét jelenti. A DOS 3 a DOS-táblában levő bármelyik országkódhoz tartozó információt tartalmazza.

Az *AL*-regisztert 0-ra kell beállítani, hogy a szabványos országinformációt kapjuk. A DOS 3.00-nál vagy a későbbi változatoknál az *AL*-regisztert egy előre meghatározott országkódra lehet beállítani (az országkód a telefonhálózatban is használt háromszámjegyes kód). Ha több, mint 255 országkódhoz szeretnénk hozzáférni, az *AL*-et be tudjuk állítani 255-re (hexa FF), az országkódot pedig betehetjük a *BX*-regiszterbe. Ha a kért országkód érvénytelen, a DOS

17.5. táblázat

A DOS 2 változatokkal nyújtott és a *DS : DX* által kijelölt 32 bájtos területen elhelyezett országfüggő információk

Mező	Eltolás	Méret, bájt	Leírás
1	0	2	A dátum időpont kódja
2	2	2	A pénzegység jele
3	4	2	Az ezres elválasztó jele
4	6	2	A tizedes elválasztó jele
5	8	24	Nem használt

beállítja a carry flaget (*CF*-et), és egy hibakódot helyez az *AX*-be. Egyébként a *BX* tartalmazza az országkódot, és a 32 bájtos terület megtelik a 17.5. és a 17.6. táblázatban bemutatott országfüggő információval. Ekkor a *DS:DX* regiszter a 32 bájtos terület elejére mutat.

A DOS 2 alatt működő programok csak a DOS-ban kódolt országspecifikus információ szabványkészletét tudják használni az *AL* 0-ra állításával. A DOS 3 alatt működő programok kérhetik azokat a (bármely országra vonatkozó) szabványinformációkat, amelyek kódja a táblázatban létezik (*AL* = kód), vagy egyszerűen megkereshetik az érvényes kódokat.

17.6. táblázat

A DOS 3 változat 56-os (hexa 38) függvényében az *AL* = országkód használatával kapott országfüggő információk

Mező	Eltolás	Méret, bájt	Leírás
1	0	2	A dátum és az időpont kódja
2	2	5	A pénzegység jele (ASCIIZ-formátum)
3	7	2	Az ezres elválasztó jele (ASCIIZ-formátum)
4	9	2	A tizedes elválasztó jele (ASCIIZ-formátum)
5	11	2	A dátum elválasztó jele (ASCIIZ-formátum)
6	13	2	Az időpont elválasztó jele (ASCIIZ-formátum)
7	15	1	A pénzegység jelének elhelyezése: (1 = az érték előtt; 0 = az érték után)
8	16	1	A pénzegység tizedes helye
9	17	1	Időformátum: 1 = 24 órás óra; 0 = 12 órás óra
10	18	4	Kisbetű és nagybetű használatát leíró rutin címe
11	22	2	A lista elválasztó jele (ASCIIZ-formátum)
12	24	8	Fenntartva

Ha a kódokban rejlő lehetőségeket szeretnénk kiaknázni, egy viszonylag egyszerű programot írhatunk, amely az összes lehetséges országkódot megvizsgálja, és kijelzi az érvényeseket. A DOS 2 változatokban a 32 bájtos információterület kódolása némileg eltér a DOS 3-étól és a későbbi változatokétól. Bár a kódolás hasonlónak tűnhet, a formátum nem kompatíbilis.

Az 1. mező (mindkét formátum) tárol egy integer szót, amelynek értéke meghatározza az idő és a dátum megjelenítési formátumát. Három előre meghatározott érték létezik erre a szóra, és három megfelelő dátum, ill. időformátum (l. a 17.7. táblázatot). A későbbiekben ez bővülhet.

17.7. táblázat
A három előre meghatározott idő- és dátumformátum, amelyek kódjait a *DS : DX* által kijelölt 32 bájtos terület első két bájtja ad meg

Érték	Használat	Idő	Dátum
0	amerikai	h : m : s	m-d-y
1	európai	h : m : s	d.m.y
2	japán	h : m : s	y-m-d

A DOS 2 formátumban a következő három mező három ASCIIZ-tringet tárol, ezek mindegyike 2 bájtos: egy adatbájt, amelyet a szabványos zéróbájt követ, ez zárja le az összes ASCIIZ-stringet.

A 2. mező az első stringet tárolja, ez adja meg a pénznem jelét. Az Egyesült Államok esetében ez a dollár jele ($).

A 3. mező a második stringet tárolja, ez adja meg a számokban levő ezres csoport jelölésére használt írásjeleket. Az USA-ban ez vessző (pl. a 12,345), más országok pontot vagy helyközt használnak.

A 4. mező a harmadik stringet tárolja, ez adja meg a tizedespont jelét. Az USA-ban ez pont (pl. 3.00), más országok vesszőt használnak.

A fennmaradó 24 bájt nincs kihasználva a DOS 2 formátumában.

A DOS 3 formátuma ugyanazzal az egész számmal kezdődik, amely a DOS 2-ben a dátum- és az időformátumot mutatja. Ezt követi ugyanaz a három ASCIIZ-string, amely egyetlen lényeges különbséggel meghatározza a jeleket a pénznemre, az ezrekre és a tizedes elválasztójelekre: a pénznemjel ASCIIZ-string számára a 2. mezőben öt bájt áll rendelkezésre. Így lehetővé teszi, hogy a pénznemjel vagy olyan rövid legyen, mint egyetlen jel (l. a dollár vagy a yen jelét), vagy négybetűs rövidítés legyen (pl. a DOS 3.00 által használt pénznemek egyike a DKR, ez a dán korona rövidítése).

Az 5. és a 6. mező ugyancsak 2 bájt hosszú az első adja meg az írásjeleket, amelyeket a dátumban használunk (pl. 7–4–1985), a második pedig az időre vonatkozó írásjeleket (pl. 12 : 34).

A 7. mező 1 bájtos, ezzel jelöljük ki, hogy a pénznem jele hová kerüljön a kimeneten. 0-nál az összeg után teszi a jelet (10 DKR); 1-nél pedig az összeg elé ($ 10.00).

A 8. mező 1 bájtos, ez határozza meg, hogy hány tizedesjel használatos a pénznemben. Az USA pénznemére ez pl. 2 lenne (dollárok és centek), az olasz pénznemre viszont 0 (líra).

A 9. mező 1 bájtos, amely bitkódolt az időformátum meghatározására. Jelenleg csak az első bit (0. bit) használatos; ha a bit = 0, 12 órás órát használunk, ha pedig a bit = 1, akkor 24 órásat.

A 10. mező egy a 8 bájtos szubrutin szegmentált címét tartalmazza, ezzel határozhatjuk meg a kisbetű és a nagybetű használatát.

A 11. mező egy 2 bájtos ASCIIZ-stringet tartalmaz, ez adja meg a listákban felsorolt tételek elválasztására szolgáló jelet (pl. a vesszőt az A, a B, a C és a D listában).

A 12. mező a 32 bájtos terület maradék 8 bájtja, amelyet a későbbi használatra tartottak fenn.

Az 57-es függvény (hexa 39): MKDIR: katalógus létrehozása

Az 57-es függvény éppúgy egy katalógust alkot, mint az **MKDIR** DOS-utasítás. E szolgáltatás meghívásakor az új katalógus útvonalnevét tartalmazó ASCIIZ-stringet, valamint az őt követő, a string lezárását biztosító zéróbájtot adjuk bemenetként. A *DS*:*DX* regiszterpár az ASCIIZ-string címére mutat. A hibákat az *AX*-szel a kiterjesztett utasítások szabványformátumában kapjuk meg. (☛ L. a fejezet hibakódokkal foglalkozó részét.) A lehetséges hibakódok a 3-as (az útvonalat nem találja) és az 5-ös (a hozzáférés megtagadva).

Az 58-as függvény (hexa 3A): RMDIR: katalógus törlése

Az 58-as függvény egy katalógust ugyanolyan módon távolít el (töröl), mint az **RMDIR** DOS-utasítás. E szolgáltatás meghívására az eltávolítandó katalógus útvonalnevét tartalmazó ASCIIZ-stringet és az azt lezáró zéróbájtot adjuk bemenetnek. A *DS*:*DX* regiszterpár az ASCIIZ-string címére mutat. A hibakódokat a kiterjesztett utasítások szabványformátumában az *AX*-en keresztül jelzi. (☛ L. a fejezet hibakódokkal foglalkozó részét.) A lehetséges hibakódok a 3-as (az útvonalat nem találja) és az 5-ös (a hozzáférés megtagadva).

A DOS nem távolítja el asem az aktuális katalógust, sem bármely másikat, amelyben alkatalógus, ill. állomány van. Ezekben az esetekben az 5-ös kód (a hozzáférés megtagadva) jelzi a hibát – legalábbis így mondja a DOS dokumentációja. A józan ész azonban azt mondja, hogy az aktuális katalógus eltávolításának kísérlete a 16-os hibakódot (az aktuális eltávolítására való kísérlet) adja.

Az 59-es függvény (hexa 3B): CHDIR: aktuális katalógus megváltoztatása

Az 59-es függvény ugyanúgy változtatja meg az aktuális katalógust mint a **CHDIR** DOS-utasítás. E szolgáltatás meghívásakor az új katalógus útvonalának a nevét tartalmazó ASCIIZ-stringet kell bevinnünk, ezt követi az ASCIIZ-stringet lezáró zéróbájt. A *DS*:*DX* a bemeneti string címét tartalmazza. A hibakódokat a kiterjesztett utasítások szabványformátumában az *AX* jelenti. Az egyetlen lehetséges hibakód a 3-as (az útvonalat nem találja).

A 60-as függvény (hexa 3C): CREAT: állomány létrehozása

A 60-as függvény vagy egy létező állományt nyit meg, vagy egy újat alkot. Ez a kimeneti állományok szabványművelete: „Keresd meg, vagy hozd létre". Ezt párhuzamba állíthatjuk a 22-es függvénnyel. (☛ L. a 16. fejezetet.)

E szolgáltatás meghívásakor egy ASCIIZ-kódot adunk, amely az útvonalnevet és az állománynevet tartalmazza, ill. a stringet lezáró zéróbájtot. A *DS* : *DX* regiszterpár az ASCIIZ-string címére mutat. A *CX* (valójában az *CL*) tartalmazza az állományattribútumot. (☛ Az állomány-attribútumot és az attribútum bitjeinek beállítását részletesebben l. a 67-es függvénynél.) Az állományfoglaltság kódját az *AX*-ben kapjuk vissza.

Ez a függvény megnyit bármilyen régi vagy új állományt olvasás vagy írás hozzáférésre, de bemeneti állományt feltételez, ezért hosszúságát 0-ra állítja.

A visszaadott kódok a 3-as, (az útvonalat nem találja), a 4-es (a fogó nem áll rendelkezésre) és az 5-ös (a hozzáférés megtagadva) lehetnek. Az 5-ös kód mutathatja, hogy vagy nincs hely az új katalógusbelépési pontra, vagy a meglevő állományt csak olvashatónak jelöltük, és nem nyithatjuk meg a kimenetnek. Mivel az *AX*-regisztert vagy az állományfogó, vagy a visszaadott kód visszaadására használjuk, csak a *CF*-jelzőbitre támaszkodhatunk a hiba jelzésénél. (☛ A további magyarázatot l. a hibakódok ismertetésénél.)

A 61-es függvény (hexa 3D): állomány megnyitása

A 61-es függvény a legáltalánosabb állománymegnyitás. Az útvonalnevet és az állománynevet ASCIIZ-string formájában adjuk meg, amelyet egy zéróbájt követ. Mint az állományra vonatkozó öszes többi I/0 függvénynél, a *DS* : *DX* erre a stringre mutat. Az *AL*-regiszterben levő kód jelöli, milyen módon kívánjuk az állományt használni. Az *AL* nyolc bitjét négy mezőre osztjuk a 17.8. táblázat szerint.

17.8. táblázat
A 61-es függvényre vonatkozó megnyitási mód meghatározására az *AL*-regiszterbe helyezett kódok

Bit 7 6 5 4 3 2 1 0	Használat
1	Az öröklés jelzőbitje (DOS 3)
. S S S	Osztott üzemmód (DOS 3)
. . . . R . . .	A későbbi használatra fenntartva (DOS 3)
. A A A	Hozzáférési kód (DOS 2 és DOS 3)

A megnyitási mód a DOS 2 változatokra egyszerű. Csak a hozzáférési biteket (☛ a 17.8. táblázatban AAA) használjuk, az összes többit 0-ra kell állítani. A három hozzáférési kód beállítását a 17.9. táblázatban láthatjuk.

A DOS 3.00 az öröklött és az osztott kódokat ugyanúgy használja, mint a hozzáférési kódokat. A DOS 3.00 kódok lényegesen bonyolultabbak, mivel számításba kell venniük az állománymegosztás nehézségeit. Ez a jellemző a DOS 3.00-val tűnt fel először.

A 7. (öröklés) **bit** jelzi, hogy a gyermekfeldolgozások (child process: az e program alatt futó félfüggetlen programok) öröklik-e vagy sem ennek az állománynak a használatát. Ha a 7. bit 0, a gyermekfeldolgozások használják az állományt. Ha a 7. bit 1, az állomány e program sajátja, és a gyermekfeldolgozások automatikusan nem férhetnek hozzá. (De bármely más programhoz hasonlóan, kérhetik magát a programot egy osztott bázison.)

17.9. táblázat
A 61-es függvény hozzáférési kódjának beállításai az *AL*-regiszterben

Bit 2 1 0	Használat
0 0 0	Csak olvasás
0 0 1	Csak írás
0 1 0	Írás és olvasás

A 4–6. bitek – az osztott üzemmód bitjei (☛ a 17.8. táblázatban SSS) – határozzák meg, hogy mi történjen, amikor ugyanazt az állományt többször kívánjuk újra megnyitni. Öt osztott mód van: a kompatibilitás mód (SSS = 000), az olvasást és írást megtiltó mód (SSS = 001), az írást megtiltó mód (SSS = 010), az olvasást megtiltó mód (SSS = 011), ill. a semmit sem tiltó mód (SSS = 100). Az osztott üzemód igen összetett, legjobb, ha a DOS Technical Reference kézikönyvre hagyjuk a magyarázatát.

A 3. bitet, amelyet a 17.8. táblázatban fenntartottnak jelöltünk, 0-ra kell beállítani. Figyeljük meg, hogy sem az állományattribútomot, sem a rekordméretet nem határozzuk meg. Ez a szolgáltatás bármely létező állományt megtalál (beleértve a rejtetteket is), és a rekordméretet 1 bájtra állítja be alapértékként.

A lehetséges visszaadott kódok a 2-es (az állományt nem találja), a 4-es (az állományt nem találja), az 5-ös (a hozzáférés megtagadva) és a 12-es (érvénytelen hozzáférési kód). Mivel ugyanazt a regisztert (*AX*) vagy a foglaltsági jelző vagy a visszaadott kód átadására használjuk, csak a *CF*-jelzőbitre támaszkodhatunk a hibajelzésnél. (☛ L. a hibakódok ismertetésénél a további magyarázatot).

A 62-es függvény (hexa 3E): az állományfoglaltság lezárása

A 62-es függvény zárja le az állományt, és söpri ki az összes *BX*-regiszterben megadott állományfogóhoz kapcsolódó állománytárakat. (A kisöprés a logikailag már lemezre kivitt, de fizikailag még fel nem írt rekordok fizikai felírását, és az átmeneti tárolók felszabadítását jelenti. A lektor.)

A *BX*-nek kell azt az állományfogót tartalmaznia, amelyet a legutolsó nyitott állományművelet visszaadott. Az egyetlen lehetséges hibakód a 6-os (érvénytelen foglaltsági jelző).

A 63-as függvény (hexa 3E): olvasás állományból vagy készülékről

A 63-as függvény olvassa a *BX*-ben megadott foglaltsági jelzővel kapcsolatban levő állományt vagy készüléket (amely állományként viselkedik). A *CX*-regiszter határozza meg az olvasandó bájtok számát, a *DS*:*DX* pedig arra a tárcímre mutat, ahová az olvasott adatot helyezzük. A függvény végrehajtása után az *AL* a ténylegesen olvasott bájtok számát tartalmazza. Ha az érték 0, a program az állomány végétől próbál olvasni.

A lehetséges hibakódok az 5-ös (a hozzáférés megtagadva) és a 6-os (érvénytelen foglaltsági jelző). Mivel ugyanazt a regisztert (*AX*-et) vagy a beolvasott bájtok számának visszaadására, vagy a visszaadott kódra használjuk, csak a *CF*-jelzőbitre támaszkodhatunk a hibajelzést illetően. (☛ A továbbiakat l. hibakódok ismertetésénél.)

A 64-es függvény (hexa 40): írás állományba vagy készülékre

A 64-es függvény a *BX*-ben levő foglaltsági jelzőhöz kapcsolt állományba vagy készülékre ír. A *CX* határozza meg az írandó bájtok számát, a *DS : DX* pedig az adatbájtok címére mutat. A függvényt végrehajtva, az *AX* tartalmazza a ténylegesen kiírt bájtok számát.

Két mód van a hibák felismerésére: az egyik a *CF* által jelzett igazi hibakód. Ha a *CF* hibát jelez, a lehetséges hibakódokat az *AX*-ben találjuk. Ezek az 5-ös (a hozzáférés megtagadva) és a 6-os (az érvénytelen foglaltsági jelző) lehetnek. Ha nem fedezünk fel nyilvánvaló hibát, de a funkció mégis hibával fut le, ez azt jelenti, hogy nem volt elég lemezterület a teljes állomány írására. Ennek elemzésére nézzük meg, hogy kevesebb bájt van-e írva (az *AX*-ben), mint amennyit igényeltünk (a *CX*-ben).

A 65-ös függvény (hexa 41): állomány törlése

A 65-ös függvény egy állomány tárgymutatóbeli belépési pontját törli. Az állományt egy ASCIIZ-string határozza meg útvonalnévvel és állománynévvel, ezt követi a záró zéróbájt. A *DS : DX* regiszterpár a stringre mutat.

A csak olvasható állományokat ezzel a függvénnyel nem lehet törölni. Ezek törléséhez az állomány attribútumát 0-ra kell állítani, a 67-es (hexa 43) függvény használatával. Globális állománynév karaktereket (a ? és a * karaktereket) nem lehet használni. Az *AX*-ben visszaadott hiba lehet a 2-es (az állományt nem találja) és az 5-ös (a hozzáférés megtagadva).

A 66-os függvény (hexa 42): az állománymutató mozgatása

A 66-os függvényt arra használjuk, hogy az állományban a logikai olvasás vagy írás helyét változtassuk. E függvény meghívásakor a *BX*-be töltjük az állományfoglaltsági jelzőt, a mutató kezdőhelyének számítási módját meghatározó kódot az *AL*-be helyezzük, a *CX : DX* regiszterekbe pedig a bájtok számát (amennyivel azt elmozgatni akarjuk). A *CX : DX* regiszterben a bájteltolás egy 32 bit hosszú előjel nélküli integer. A *CX* az eltolás magas értékű része (amely 0, hacsak az eltolás összege nincs 65 535 felett), a *DX* pedig az alacsony értékű rész.

Az *AL* tartalmát metóduskódnak hívjuk, és három értéket vehet fel. Ha az $AL = 0$, akkor az eltolást az állomány kezdetétől vesszük, és a mutató *CX : DX* bájtnyit mozdul el; ha $AL = 1$, akkor az aktuális helytől; ha pedig $AL = 2$, akkor az aktuális állomány végétől számítjuk az elmozdulást. Az utolsó esetben az eltolást rendszerint 0-ra állítjuk azért, hogy kitaláljuk az állomány aktuális méretét. Ha ezt az eltolást 0-ra állítjuk, és a 0 metódust kérjük, akkor visszatérünk az állomány elejére.

A függvény visszatéréskor a *DX : AX* regiszterekben adja meg, hogy az állománymutató az állomány elejétől hány bájtnyira mutat. A mutatót 32 bit hosszúságú integerként kapjuk vissza, ahol a magasabb értékű rész a *DX*-ben, az alacsonyabb pedig az *AX*-ben van. Vegyük

észre a regiszterpár furcsa megválasztását! A lehetséges hibakód az 1-es (érvénytelen függvényszám, amely az eljárási kód alfüggvényre vonatkozik), és a 6-os (érvénytelen foglaltsági jelző). Mivel az új hely részeinek és a visszaadási kódnak a visszaadására is ugyanaz a regiszter (*AX*) használatos, mégis a *CF*-jelzőbitre támaszkodunk. (☛ L. a fejezet hibakódokat ismertető részét.)

A 67-es függvény (hexa 43): CHMOD: állományattribútumok lekérdezése vagy beállítása

A 67-es függvény kérdezi le vagy állítja be egy állomány attribútumait. (☛ Az állományattribútumokról a részleteket l. az 5. fejezetben). A *DS:DX* egy ASXIIZ-stringre mutat, amely a kérdéses állomány specifikációját adja (globális állománynév-karaktereket – *-ot és ?-et nem használhatunk). Az *AL*=0 esetén a *CX*-ben kapjuk meg a katalógusban talált attribútumértékeket; az *AL*=1 pedig a *CX*-ben talált attribútumértékeket állítja be. Mindkét esetben valójában a *CL* regiszterből vesszük az értékeket.

A lehetséges hibakód az *AX*-ben: a 2-es (az állományt nem találja), a 3-as (az útvonalat nem találja) és az 5-ös (a hozzáférés megtagadva).

A 68-as függvény (hexa 44): IOCTL: I/0 berendezések vezérlése

A 68-as függvény számos bemenet- és kimenetvezérlési műveletet hajt végre, főleg olyan berendezéseken, amelyek mindnyájan egy összekuszált csomagban vannak összegyűjtve. (L. a 17.10. táblázatot.) Az *AL* választja ki a 0-tól 8-ig, ill. 11-gyel számozott tíz alfüggvény egyikét. (A 8-as és 11-es alfüggvények a DOS 3 változatokra vonatkoznak.) A *BX* adja vissza az állományfoglaltsági jelzőt.

A 0-ás és az 1-es alfüggvény: ezek az alfüggvények kérdezik le és állítják be a berendezés információit, amelyek egy bonyolult bitkódolási készletként állnak össze a *DX*-ben. A 7. bit

17.10. táblázat
A 68-as függvénnyel nyújtott tíz I/O vezérlő utasítás (kiválasztás az *AL*-regiszter révén)

Kód	Leírás
0	A készülékinformáció lekérdezése (a *DX*-ben a válasz)
1	A készülékinformáció beállítása (a *DH*-nak 0-nak kell lennie)
2	Olvasás (l. a megjegyzéseket)
3	Írás (l. a megjegyzéseket)
4	Olvasás a lemezegységről (l. a megjegyzéseket)
5	Írás a lemezegységre (l. a megjegyzéseket)
6	A bemeneti állapot lekérdezése (l. a megjegyzéseket)
7	A kimeneti állapot lekérdezése (l. a megjegyzéseket)
8	Ha a készülék cserélhető; csak a DOS 3 változatoknál (l. a megjegyzéseket)
11	Osztott bemenet megváltoztatása; csak a DOS 3 változatoknál (l. a megjegyzéseket)

a berendezések esetén 1-re állítódik, a lemezállományokra pedig 0-ra. A berendezésekhez használatos 0–5. bitet a 17.11. táblázat mutatja be. A lemezállománynál a 0–5. bit a lemezegység számát adja: a 0 képviseli az *A* lemezegységet, az 1 a *B* lemezegységet és így tovább.

A 2–5--ös alfüggvény: mind a négy függvényben a *CX* határozza meg azoknak a bájtoknak a számát, amelyeket a lemez vezérlő csatornájáról olvasunk vagy arra írunk (nem egy állományra a lemezegységen belül), a *DS*:*DX* regiszterpár pedig az adatterületre mutat. A 4. és az 5. alfüggvényeknél a *BL* tartalmazza a speciális lemezegységszámokat (0 = alap, 1 = *A*, 2 = *B* stb.).

E függvények használatával adatokat csak akkor olvashatunk és írhatunk, ha a készülék vagy a lemezegység fel tudja dolgozni a vezérlő stringeket. Az írás/olvasás állapotot a *DX* állapotbitjei közül a 14. bit mutatja.

A 6-os és 7-es alfüggvény: e függvényekben a bemeneti és a kimeneti állapotot az *AX*-regiszter jelzi, ha *AX* = 255 (hexa FF), a berendezés készen áll I/0-ra; ha *AX* = 0, akkor nem.

A 8-as alfüggvény. ez a függvény csak a DOS 3-ra és a későbbi változatokra vonatkozik. Azt mutatja meg, hogy egy készüléknek cserélhető vagy nem cserélhető közege van-e (a hajlékonylemezes egységben levő lemez cserélhető; a Winchester-lemezegységben levő lemez viszont nem). A 8-as alfüggvény rendkívül hasznos lehet, mivel vele a program megismerheti,

17.11. táblázat

A 68-as függvénynek a 0 vagy az 1 alfüggvényre vonatkozó *DX*-regiszterbeli bitbeállítása

Bit F E D C B A 9 8 7 6 5 4 3 2 1 0	Használat
. X	1 = szabvány konzolbemenet
. X .	1 = szabvány konzolkimenet
. X . .	1 = null készülék (nem kell I/O-t végeznie)
. X . . .	1 = órakészülék
. X	1 = speciális készülék (bármit is jelezzen ez)
. X	1 = adat „nyersen” kerül továbbadásra; 0 = az adat feldolgozottan („főzve”) kerül továbbadásra
. X	0 = állomány vége; 1 = nincs vége az állománynak (leolvasás esetén)
. X	1 = készülék; 0 = lemezegység-állomány
. R 	Fenntartva
. R	Fenntartva
. R	Fenntartva
. . . . R	Fenntartva
. . . R	Fenntartva
. . R	Fenntartva
. X	1 = a készülék olvasás/írás alatt nem tud vezérlőstringeket feldolgozni
R	Fenntartva

hogy kell-e a lemez változtatását ellenőrizni, vagy bízhat abban, hogy ugyanaz a lemez mindig ugyanott marad. A cserélhető készüléket az $AX=0$ jelzi, a rögzített lemezegységet az $AX=1$, az érvénytelen készülékkódot pedig az $AX=15$ (hexa F).

A 11-as alfüggvény (hexa B): ez az alfüggvény csak a DOS 3-ra és a későbbi változatokra vonatkozik. Arra használjuk, hogy vezérelje azokat a kísérleteket, amelyek az állománymegosztásból származó konfliktusokat kívánják megoldani. Az állománymegosztás nehézségei percről percre változhatnak, hiszen néhány program az állományokat csupán rövid ideig használja. A DOS mielőtt konfliktust jelez, egy osztott állományhoz egynél többször is megkísérelhet hozzáférni, abban a reményben, hogy időközben a hozzáférés letiltása elmúlt. Ez a függvény állítja be a DOS próbálkozásainak számát – ezt a *DX*-ben határozzuk meg –, és beállítja a próbálkozások közti időintervallumot is (néhány nem meghatározott egységben), amelyet egy számértékkel a *CX*-regiszterben jelölünk.

A 69-es függvény (hexa 45): DUP: állományfoglaltság másolása

A 69-es függvény egy nyitott foglaltsági jelzőt másol le, és egy új foglaltsági kódot ad vissza, amely ugyanarra az állományra vagy berendezésre vonatkozik. Az egyik végrehajtott összes művelet automatikusan tükröződik a másikban is. Az új jelző semmi esetre sem változhat a másiktól függetlenül. (☛ L. a 70-es függvény idevonatkozó szolgáltatását.)

A *BX*-regiszter tartalmazza a régi állományfoglaltságot az *AX* pedig visszaadja az újat. A lehetséges hibakód a 4-es (foglaltsági jelző nem áll rendelkezésünkre) és a 6-os (érvénytelen foglaltsági jelző).

Eddig nem találtuk meg e függvény hasznosításának módját, de biztos, hogy van ilyen.

A 70-es függvény (hexa 46): CDUP a foglaltság másolásának kikényszerítése

A 70-es függvény a 69-es függvényhez hasonlóan egy foglaltsági jelzőt másol, ebben az esetben azonban egy létező másodikat kapunk, amelyet ezt megelőzően feltehetően más célra használtunk (ahelyett, hogy a DOS alkotna egy újat). Ha a második jelző egy nyitott állományra hivatkozik, az állományt lezárja, mielőtt bármilyen tevékenység megtörténne. Ha ezt a függvényt már kiváltottuk, az egyik állománnyal végrehajtott összes tevékenység automatikusan tükröződik a másikban is. Az új foglaltsági jelző semmi esetre sem független a régitől.

A *BX*-regiszter tartalmazza a régi foglaltsági jelzőt a *CX* pedig a másodikat. Amikor a művelet befejeződik, a *CX*-ben levő foglaltság ugyanarra a készülékre vagy állományra vonatkozik, mint a *BX*. Az egyetlen lehetséges hibakód a 6-os (érvénytelen foglaltsági jelző). Ez a szolgáltatás ugyan elég különösnek tűnhet, mégis van néhány ésszerű felhasználása. Pl. egy program vele átirányíthat bármilyen szabvány I/0 készüléket. (☛ A leírást és a listát l. a 68-as függvény ismertetésénél.) Feltételezzük, hogy a program dinamikusan át akarja irányítani a nyomtatókimenetet egy állományra. Először az állományt megnyitja, és egy fogót ad vissza a *BX*-ben; majd ez a foglaltsági jelző duplikálódhat a szabványnyomtató foglaltsági jelzőjével a 4-es jelzőszám *CX*-be való töltésével. Ezután bármely, a szabványos nyomtatókimenetre szánt írás ténylegesen a megnyitott állományra megy. Ha a rendes nyomtatóirányt vissza akarjuk állítani, mert egy későbbi használatra a jelzőt szeretnénk megtartani, akkor a szabványnyomta-

tó jelzőjét egy másik foglaltsági jelzőbe másolással ki kell mentenünk, mielőtt az új állományfogót a nyomtatófogóval másolnánk. Ha követjük ezt az utasítást, akkor elég képzettek vagyunk e függvény kipróbálására. (☛ E technika egyik felhasználására l. a 75-ös (hexa 4B) függvényt.)

A 71-es függvény (hexa 47): az aktuális katalógus lekérdezése

A 71-es függvény jelenti az aktuális katalógust egy ASCIIZ-string formájában. A lemezegység számát a *DL*-ben határozzuk meg (0 = alap, 1 = *A* lemezegység, 2 = *B* lemezegység). A *DS* : *SI* regiszterpár egy olyan adatterületre mutat, amely tartalmazza a teljes útvonalnevet, ennek hossza 64 bájt lehet. A DOS a meghatározott lemezegységre vonatkozó aktuális katalógus teljes útvonalnevét visszaadja, belefoglalva a gyökérkatalógust is. Az útvonalnevet mindig egy zéróbájt – az ASCIIZ-stringet lezáró – követi.

Bár ez a függvény a teljes útvonalnevet adja vissza, a név nem foglalja magába sem a lemezegység azonosítóját (mint az **A**:-ban), sem a gyökértől való indulást (mint az **A**:\-ban). E szabállyal, ha az aktuális katalógus a gyökérkatalógus (ez mindennapi jelenség a hajlékonylemezeken), akkor az aktuális tárgymutató *semmi*ként – azaz nullstringként – jelentődik. Ha az aktuális katalógust intelligens módon akarjuk megjeleníteni, akkor a lemezegység- és útvonaljelzőket (éppúgy, mint az **A**:\-ban) az e függvény által visszaadott információ elé kell tenni. Az egyetlen lehetséges hibakód a 15-ös (érvénytelen lemezegység).

A 72-es függvény (hexa 48): memória lefoglalása

A 72-es függvény dinamikusan utalja ki a memóriát. A *BX*-ben adjuk meg azoknak a bekezdéseknek a számát (ezek 16 bájtos egységek), amelyeket szeretnénk lefoglalni. Visszatéréskor az *AX* a kiutalt memóriatömb szegmensbekezdésére mutat.

A lehetséges hibakód a 7-es (a memóriavezérlő tömbök tönkrementek) és a 8-as (nem elegendő a memória). Ha a függvény nem utal ki memóriát, a *BX*-regiszter az elérhető legnagyobb tömb méretét adja vissza.

A 73-as függvény (hexa 49): lefoglalt memória felszabadítása

A 73-as függvény adja vissza a DOS-nak azt a memóriát, amelyet a 72-es függvény utalt ki. Az *ES*-regiszter a visszaadott tömb szegmensére mutat. Ez ugyanaz az érték, amelyet a 72-es (hexa 48) függvény ad vissza az *AX*-regiszterben.

A lehetséges hibakód a 7-es (a memóriavezérlő tömbök tönkrementek) és a 9-es (érvénytelen memóriatömbcím).

A 74-es függvény (hexa 4A): SETBLOCK: a lefoglalt memóriatömb méreteinek módosítása

A 74-es függvényt használjuk arra, hogy a 72-es függvénnyel kiutalt memóriatömbök méretét növeljük vagy csökkentsük. Az *ES*-regiszter mutat a változtatandó tömb szegmens címére. A *BX*-regiszter bekezdésekben (16 bájtos egységekben) tartalmazza a tömb új méretét.

Ha egy terület növelésének igénye nem teljesíthető, akor a *BX* a legnagyobb elérhető memóriatömb méretét adja vissza (ugyancsak bekezdésekben mérve). A lehetséges visszaadott kódok: a 7-es (a memóriavezérlő tömbök tönkrementek), a 8-as (nem elegendő a memória) és a 9-es (érvénytelen memóriatömbcím).

A 75-ös függvény (hexa 4B): EXEC: program betöltése és végrehajtása

Az EXEC-függvény lehetővé teszi egy program számára, hogy a memóriába alprogramot töltsön be, és opcionálisan végrehajtsa. A *DS : DX* regiszterpár a betöltendő állománynevet és útvonalnevet tartalmazó ASCIIZ-stringre mutat. Az *ES : BX* regiszterpár arra a paramétertömbre mutat, amelyik tartalmazza a betöltési műveletek vezérlő információit. Az *AL* határozza meg azt, hogy betöltése után az alprogram végrehajtandó-e.

Ha az $AL=0$, az alprogram betöltődik, a programszegmens előtagja (PSP) létrejön, és a program végrehajtódik. Ebben az időpontban a vezérlés átkerül az alprogramra, és csak akkor tér vissza a programba, ha az alprogram véget ér. Ha az $AL=3$, az alprogram betöltődik, de nem jön létre a PSP, és a program sem jut le automatikusan (bár rá tudunk ugrani). Az $AL=3$ változatot rendszerint a programátlapolás (overlay) betöltésére használjuk. Ugyancsak egyszerű és hatékony mód adatok memóriába töltésére is.

Ha az $AL=0$, a tömb (amelyre az *ES : BX* mutat) 14 bájt hosszú, és a 17.12. táblázatban bemutatott információt tartalmazza. Ha az $AL=3$, a tömb 4 bájtos, és a 17.13. táblázatban bemutatott információt tartalmazza.

17.12. táblázat
AL = 0 esetén az EXEC vezérlőblokkban levő információ (*ES : BX* mutat rá). E tételek mindegyike a betöltendő program PSP-jébe beépített információkra vonatkozik

Eltolás	Méret, bájt	Megnevezés
0	2	A környezetet leíró string szegmenscíme
2	4	Szegmentált mutató az utasítás sorára
6	4	Szegmentált mutató az első FCD-re
10	4	Szegmentált mutató a második FCB-re

17.13. táblázat
Az *AL* = 1 értéknél az EXEC vezérlőblokkban lévő információ (az *ES:BX* mutat rá)

Eltolás	Méret, bájt	Megnevezés
0	2	A szegmenscím, ahová az állományt tölteni kell
2	2	A program relokációs tényezője (csak EXE formátumú programokra vonatkozik)

Ha egy program betöltődött és végrehajtásra kerül, számára bármelyik (pillanatnyilag aktív) állományfogó elérhető. A 70-es (hexa 46) függvénynél már említettük, hogy egy program átirányíthatja a szabványos I/0 fogókat, és ezt a módszert egy alprogram működésének befolyásolására használhatja. Előidézhetünk pl. egy szabványos rendező-, ill. szűrőprogramot egy állománykészlet rendezésére; és annak kimenetét, üzeneteit vagy hibaüzeneteit bárhol hagyhatjuk, ahová azokat küldeni akarjuk.

E függvény használata előtt meg kell győződnünk a 74-es (hexa 4A) függvény alkalmazásával, hogy a program betöltéséhez elegendő memória áll rendelkezésre. Mivel a programbetöltési eljárást az utasításértelmező (command interpreter) hajtja végre, az értelmező fél-rezidens részének érintetlennek kell lennie, vagy újra be kell tölteni a lemezről, ha megváltozott.

A DOS utasításértelmezője azok között a programok között van, amelyeket betölthetünk és végrehajthatunk. Ha kívánjuk, akár be is tölthetjük, és adhatunk neki egy utasításstringet is, ami viszont futtathat egy kötegelt állományt, egy olyan kötegelt állományt, amelyet az eredeti programunk dinamikusan alkothatott meg. Ez a kötegelt állomány is futtathatja a programjait, végrehajtva egy EXIT-utasítást, amely a másodlagos utasításértelmező végrehajtását fejezi be. Ennél a pontnál az eredeti programunk újra visszavenné a vezérlést. Ez a tulajdonság hatalmas és összetett lehetőségeket tár fel.

✱ FIGYELMEZTETÉS: *A betöltés és végrehajtás függvény belekontárkodik a legtöbb regiszterbe, beleértve a veremregisztereket, az SS-t és az SP-t is. E függvény meghívása előtt ezeket el kell menteni, és utána vissza kell állítani. Nyilvánvaló, hogy a vermet nem lehet erre használni. A probléma egyik csúnya megoldása az, ha a veremregisztereket a programkódon belül tároljuk. Ha ezt tesszük, a szolgáltatásból való visszatérés után az első néhány végrehajtott utasítást arra kell szentelni, hogy a szükséges regisztereket visszaállítsa. Ezek az utasítások közvetlenül a regiszterekbe visznek értékeket: épp a regisztereknek e szolgáltatás meghívása előtt elmentett értékeit. Az ilyenfajta programozási technikát – a programutasítások közvetlen módosítását – a jó programozók valójában elítélendőnek tartják, de néha ezt a bűnt is el kell követni.*

A lehetséges visszaadott kódok az 1-es (érvénytelen függvényszám), a 2-es (az állományt nem találja), az 5-ös (a hozzáférés megtagadva), a 8-as (nem elegendő a memória), a 10-es (érvénytelen környezet) és a 11-es (érvénytelen formátum).

A 76-os függvény (hexa 4C): a folyamat befejezése

A 76-os függvény befejez egy programot és egy kódot küld vissza. Ha a programot alprogramként hívtuk meg, a visszaadott kódot a 77-es függvénnyel találjuk meg; ha pedig DOS-utasításként, akkor a visszaadott kódot a DOS ERROR LEVEL utasításának kötegelt állományban való használatával lehet vizsgálni. Ekkor a visszaadott kód az *AL*-regiszterben jelenik meg.

Amikor ezt a függvényt végrehajtjuk, a DOS automatikusan lezár minden állományt, amelyet a 61-es (hexa 3D) függvénnyel (és feltehetően a 60-assal is) nyitottunk meg.

A 77-es függvény (hexa 4D): az alprogram visszatérési kódjának lekérdezése

A 77-es függvény olvassa ki a meghívott és a 76-os függvénnyel befejezett alprogram visszaadott kódját. A visszaadott információ két részből áll: az *AL* jelenti a program által visszaadott kódot, az *AH* pedig azt, hogyan fejeződött be a program. Négy lehetséges értéke lehet: az *AH*=0 egy normál (önkéntes) befejezést jelez; az *AH*=1 a *Ctrl-Break* következtében beállott DOS-befejezést; az *AH*=2 egy kritikus készülékhiba következtében a DOS által végzett befejezést; az *AH*=3 pedig a 49-es (hexa 31) függvény használatával végrehajtott önkéntes befejezést.

A 78-as függvény (hexa 4E): FIND FIRST: az állománykeresés indítása

A 78-as függvény keresi az első illeszkedő állományt – vagy illeszkedő állományokat –, amelyek megfelelnek az állományleírásnak. A *DS:DX* egy olyan ASCIIZ-stringre mutat, amely megadja a keresett útvonalnevet és állománynevet. Az állománynév tartalmazhat globális állománynév-karaktereket is (? és *). A *CX*-regiszter (valójában a *CL*) adja meg azt az állományattribútum specifikációt, amelyet az állománykereséshez használni fogunk. Ha megtaláltuk az állományt, a DOS kialakít egy 43 bájtos információt erről az aktuális lemezátvitel területén (DTA). (L. a 17.14. táblázatot.)

Az információs terület végén levő ASCIIZ-string az állománynevet annak hagyományos formátumú jelzésével tárolja, beleértve az állománynév és a kiterjesztés közti pontot is. Ha a kiterjesztés üres, a pont nem jelenik meg.

Ez a szolgáltatás hasonló a hagyományos 17-es (hexa 11) DOS-függvényhez. Az állományattribútumok használata ebben a keresési függvényben ugyanaz, mint a 17-es függvényben. (☛ L. a 16. fejezetet.)

Az attribútum keresése egyedi logikát követ. Ha rejtett-, rendszer- vagy katalógus állomány attribútumbitjeinek kombinációját határozzuk meg, a keresés illeszteni fogja a rendes állományokat, és az ilyen attribútumokkal bíró állományokat is. Ha kötetcímke (volume label)

17.14. táblázat

A 78-as (hexa 4E) függvény hívása után a DTA-ban visszaadott információ

Eltolás	Méret, bájt	Megnevezés
0	21	A területet a DOS a „Találd meg a következőt" függvényhez (l. a 79-es függvényt) használja
21	1	A fellelt állomány attribútuma
22	2	Az állomány időbélyegzője
24	2	Az állomány dátumbélyegzője
26	4	Az állomány mérete bájtokban
30	13	Az állománynév és a kiterjesztés (ASCIIZ-string)

attribútumát határozzuk meg, akkor a keresés csak az ilyen attribútummal ellátott katalógus-belépési pontokat illeszti. Az archív és a csak olvasható bitekre nem vonatkoznak ezek a keresési műveletek. A katalógus, a kötetcímke, az archív és a csak olvasható attribútumai a DOS 2.00 előtti változatokban mincsenek meg.

A hibakódokat szokásosan az *AX*-ben találjuk: a 2-es (az állományt nem találja) és a 18-as (nincs több fellelhető állomány). A *CF* nem jelzi a hibát.

A 79-es függvény (hexa 4F): FIND NEXT: az állománykeresés folytatása

A 79-es függvény folytatja a 78-as függvénnyel megkezdett állománykeresést. A DTA elején létrehozott információra támaszkodik, ezt nem szabad megzavarni.

Az egyetlen rendes hibakód a 18-as (a több adatállomány nincs), a *CF* nem jelzi a hibát. (☛ L. a 78-as függvényt is.)

A 84-es függvény (hexa 54): az ellenőrző állapot lekérdezése

A 84-es függvény közli velünk az ellenőrzési kapcsoló aktuális állapotát. Ez vezérli, hogy a lemezre írás műveletei ellenőrzöttek-e vagy nem. Az $AL=0$ azt jelzi, hogy azok nem ellenőrzöttek; az $AL=1$ pedig azt, hogy ellenőrzöttek. A 46-os (hexa 2E) függvény állítja be az ellenőrzési kapcsolót. (☛ L. a 16. fejezetet.)

Ez a függvény egy bosszantó következetlenséget okoz a DOS-szolgáltatásokban. Amíg néhány „kérdezd és állítsd be” szolgáltatáspár egy függvényben integrálódik (hasonlóan a következő 87-es függvényhez), mások két külön funkcióra oszlanak (mint a 84-es és a 46-os függvény).

A 86-os függvény (hexa 56): állományátnevezés

A szabvány DOS **RENAME**-utasításához hasonlóan a 86-os függvény egy állomány nevét változtatja meg, és egy állomány katalógus belépési pontját egyik katalógusból a másikba is át tudja vinni. Az állomány maga nem mozog, csak a belépési pont. Ez azt jelenti, hogy az új és a régi útvonalaknak ugyanazon a lemezen kell lenniük. Ez valóban csodálatos és hasznos tulajdonság, de bosszantó, hogy nem része a **RENAME**-utasításnak.

A függvénynek két bemenetre van szüksége: a régi és az új állománynév állományleírójára. Ez lehet egy teljes állományleírás is lemezegység és útvonal elemekkel. A kiválasztott és az új lemezegységeknek azonosnak kell lennie, hogy az új katalógusbejegyzés az állománnyal egy lemezegységen legyen. A globális állománynév-karaktereket (* és ?) nem használhatjuk, mivel ez a függvény csak egyedi állományokon működik.

Szokás szerint mindkét állományleírót ASCIIZ-string formájában, a végét lezáró zérőbájttal adjuk meg. A *DS*:*DX* regiszterpár a régi névre mutat, az *ES*:*DI* pedig az újra.

A lehetséges hibakód: a 3-as (az útvonalat nem találja), az 5-ös (a hozzáférés megtagadva) és a 17-es (az útvonalat nem találja).

A 87-es függvény (hexa 57): az állomány dátumának és időpontjának lekérdezése vagy beállítása

A 87-es függvény kérdezi le vagy állítja be az állomány dátumát és időpontját. Emlékezzünk vissza, hogy az összes állományt megjelölték a létrehozás, vagy az utolsó változtatás dátumával és időpontjával. Az *AL*-et a művelet kiválasztására használjuk: az *AL* = 0 lekérdezi az időt, az *AL* = 1 pedig beállítja.

Az állományt az állományfogó *BX*-be helyezésével választjuk ki, ez azt jelenti, hogy a szolgáltatás csak olyan állományokra vonatkozik, amelyeket előzőleg az e fejezetben tárgyalt kiterjesztett DOS-függvényekkel már megnyitottunk. Jegyezzük meg ezért, hogy az állomány időbélyegzőjének beállítása ezzel a szolgáltatással csak akkor hatásos, ha az állományt sikeresen zárjuk le.

A dátumot és az időt a *CX*-, ill. a *DX*-regiszterekben helyezzük el úgy, ahogy azokat a lemezegység katalógus bejegyzésében tároltuk, csak egy kissé eltérő sorrendben. Ennél a függvénynél az idő a *CX*-ben, a dátum pedig a *DX*-ben található.

A DOS dokumentációban találhatóval ellentétben, az idő- és a dátuminformációt a konvencionális formátumban helyezzük el, a magas helyiértékű részeket a *CH*-ban vagy a *DH*-ban, az alacsony helyiértékűeket pedig a *CL*-ben vagy a *DL*-ben.

A dátumot és az időt fel lehet építeni vagy le lehet bontani a következő képlet használatával:

```
CX = HOUR * 2048 + MINUTE * 32 + SECOND/2
DX = (YEAR - 1980) * 512 + MONTH * 32 + DAY
```

E szolgáltatás lehetséges hibakódjai: az 1-es (érvénytelen függvényszám – vagyis az *AL*-ben kiválasztott alfüggvény) és a 6-os (érvénytelen foglaltsági jelző).

DOS 3.00 Bővítések

Eddig azokat az új DOS-függvényeket tárgyaltuk, amelyeket a DOS 2.00-val vezettek be. A DOS 3.00 kibővített néhány DOS 2-funkciót, és 5 új függvényt hozott a kiterjesztett függvények családjába.

A 89-es függvény (hexa 59): kiterjesztett hibakód lekérdezése

A 89-es függvényt egy hiba megjelenése után használjuk. Ez részletes információt ad azokról a hibákról, amelyek az alábbi körülmények között keletkezhetnek: a kritikus hiba miatti megszakítás kezelőjén belül, ha a szabvány 33-as (hexa 21) megszakítással meghívott DOS-függvény hivatása után a carry flag (*CF*) beállításával hibát jelez; és a régi típusú FCB állományműveletek után, amely egy 255 (hexa FF) visszadott kódot jelent. Ez azonban az olyan DOS-függvényekkel nem működik, amelyek – bár hibában végződhetnek – a hibákat nem a *CF*-ben jelzik.

Ezt a függvényt a szabványos módon – a függvénykódot, a 89-et (hexa 59-et), az *AH*-regiszterbe helyezve – hívjuk. Ha az ilyen függvényekben bekövetkező elkerülhetetlen változásokat lehetővé akarjuk tenni, meg kell határoznunk a változatkódot a *BX*-regiszterben. A DOS 3.00-nál ezt a kódot 0-ra állítjuk be.

17.15. táblázat
A 89-es (hexa 59) függvény végrehajtását követően az *AH*-regiszterben visszaadott kiterjesztett hibakódok

Kód	Jelentés
1	Érvénytelen függvényszám
2	Az állományt nem találja
3	Az útvonalat nem találja
4	Nincs foglaltságjelző
5	A hozzáférés tiltva (pl. írási kísérlet csak olvasható állományra)
6	Érvénytelen foglaltságjelző
7	A memóriavezérlő-tömbök érvénytelenek
8	Nincs elég memória
9	Érvénytelen memóriatömbcím
10	Érvénytelen **SET**-utasításstringek („környezet")
11	Érvénytelen formátum (amiről nem értesültünk)
12	Érvénytelen állomány-hozzáférési kód
13	Érvénytelen adat
14	Fenntartva
15	Érvénytelen a lemezegység leírása
16	Az aktuális tárgymutató törlésének kísérlete
17	Nem ugyanaz a készülék
18	Nem találhatók további állományok
19	A lemez írásvédett
20	Nem ismert egységazonosító
21	A lemezegység nem áll készenlétben
22	Az utastás nincs meghatározva
23	Lemezadat hiba
24	Rossz az igénystruktúra hossza
25	Lemez keresési hiba
26	Ismeretlen lemez típus
27	A lemezszektort nem találja
28	Papírhiba a nyomtatónál
29	Íráshiba
30	Olvasáshiba
31	Általános hiba
32	Az állománymegosztás megsértése
33	Az állománylezárás megsértése
34	Helytelen lemezcsere
35	Az FCB nem áll rendelkezésre
80	Az állomány már létezik
81	Fenntartva
82	Nem tudja végrehajtani
83	Sikertelen a kritikushiba kezelése

E szolgáltatás befejezésénél négy különböző információt kaphatunk vissza: az *AX* tartalmazza a kiterjesztett hibakódot; a *BH* mutatja a hibaosztályt; a *BL* adja meg azokat a javasolt tevékenységeket, amelyeket a programoknak követni tanácsos; a *CH* pedig egy helykódot ad, ez megkísérli megmutatni a hiba helyét.

Az *AX*-ben jelentett hibakódok három csoportba szervezettek: az 1–18-as kódot használjuk a függvényhívás hibákra (a 33-as megszakítás függvényei), a 19–33-as kódot a kritikushiba-kezelő hibákra (a 36-os megszakítás), a 32–83-as kódot pedig azokra a hibákra, amelyek újak a DOS 3.00 szolgáltatásoknál. A 0 kód jelzi, hogy nincs jelenteni való hiba ebben a szolgáltatásban.

A 17.15. táblázat sorolja fel a kiterjesztett hibakódokat, a 17.16. táblázat a hibaosztályokat, a 17.17. táblázat a tevékenység kódokat, a 17.18. táblázat pedig a helykódokat.

17.16. táblázat
A 89-es (hexa 59) függvény végrehajtását követően a *BH*-regiszterben visszaadott hibakódok

Kód	Jelentés
1	Kifogyott a forrásból: nincs több abból, amit kértünk, bármi is legyen az
2	Időleges helyzet: próbáld újra később!
3	Meghatalmazás: számunkra nem megengedett, de valaki más számára az lehet
4	Belső hiba a DOS-ba: nem a mi hibánk
5	Hardverhiba
6	A rendszerszoftver hibája: más DOS-problémák
7	A felhasználói szoftver hibája: nem a mi hibánk
8	A kért tételt nem találja
9	Rossz formátum (pl. nem felismerhető lemez)
10	A tétel zárva
11	Közeghiba (pl. a lemez CRC-hibát jelent)
12	Már létezik
13	A hiba osztálya nem ismert

17.17. táblázat
A 89-es (hexa 59) függvény végrehajtását követően a *BL*-regiszterben visszaadott javasolt tevékenységek kódjai

Kód	Jelentés
1	Próbáld most újra!
2	Várakozás után később próbáld újra!
3	Kérd a felhasználót, hogy javítsa ki (pl. cserélje a lemezt), l. a 7. kódot is
4	Állítsd le azonnal a programot, és próbáld eltakarítani (zárd le az állományokat stb.)
5	Állítsd le azonnal: ne próbáld eltakarítani!
6	Hagyd figyelmen kívül a hibát: nem érdekes!
7	Felhasználói tevékenység után próbáld újra, l. a 3. kódot is

17.18. táblázat
A 89-es (hexa 59) függvény végrehajtását követően a *CH*-regiszterben visszaadott helykódok

Kód	Jelentés
1	Sajnos nem ismert
2	A tömbkészülék hibája (pl. lemezegység)
3	Fenntartva
4	A soros készülék hibája (pl. nyomtató)
5	Memóriahiba

A 90-es függvény (hexa 5A): időleges állomány létrehozása

A 90-es függvény időleges használatra alkot egy állományt, miközben gondoskodik az állománynév-keresés olyan aprólékos munkájáról, amely nem ütközik semmilyen már létező állománnyal. Két paramétert adunk meg: az állományattribútumot, amelyet a *CX*-regiszterbe helyezünk, és annak a katalógusnak az útvonalnevét, ahol az állomány létrejön. Ha nem akarunk külön útvonalat megadni, adjunk a DOS-nak egy null-stringet, amely közli vele, hogy az aktuális lemezegység aktuális katalógusát használja.

Az útvonalnévnek egy ASCIIZ-stringnek kell lennie, és a *DS* : *DX* regiszterpár mutat rá. Az útvonalnév stringjének alkalmasnak kell lennie, hogy a megalkotott állomány nevét hozzátoldjuk. Ezért a stringnek fordított perjelkarakterrel (\) kell végződnie, amelyet a katalógus írásjeleként használunk az útvonalon (ha kifejezetten útvonalstringet adunk meg). Hozzá kell adnunk 12 bájtot is, így elég helyet biztosítunk a DOS-nak, hogy az állománynevet hozzátoldja a stringhez.

Visszatérés után, ha hiba van a műveletben, a carry flag (*CF*) beállítódik, és a hibakód az *AX*-ben lesz. Az állománynév újra hozzáadódik a megadott útvonalstringhez.

Ez a szolgáltatás az „ideiglenes állomány létrehozása”, ez jelzi is a célját. Valójában ez az állomány, mivel a DOS azt nem törli automatikusan: ezt a munkát a programjainknak kell ellátni.

A 91-es függvény (hexa 5B): új állomány létrehozása

A 91-es függvény hasonló a 60. (hexa 3C) függvényhez, amelyet (pontatlanul) állomány-létrehozó függvénynek hívunk. A 60-as függvényt valójában egy állomány megtalálására és megalkotására tervezték, ha a kért állomány nem létezik. Ellentétben ezzel, a 91-es függvény egy tiszta állomány-létrehozó függvény, amely meghiúsul, ha az állomány már létezik.

Ugyanúgy, mint a 60-as függvénynél, a *CX*-regiszter felveszi az állományattribútumot, a *DS* : *DX* pedig az útvonalnév- és állománynévcímre mutat, amelyet egy ASCIIZ-stringként tárolunk. Visszatéréskor a *CF* jelzi a hibákat, és az *AX* adja vissza a hibakódokat.

Gyakran előfordul, hogy a program szabványos állománynevet használ és újra felhasznál egy állományt ugyanazzal a névvel, ha az létezik, vagy megalkot egy új állományt ugyanazzal a névvel, ha az nem létezik. Erre a 60-as függvény a legalkalmasabb. Lehetnek más körülmények is, amikor a program nem akarja zavarni a létező állományokat, és csak akkor akar egy állományt megnyitni, ha az még nem létezik. Erre a 91-es függvény a legalkalmasabb.

A 92-es függvény (hexa 5C): állományhozzáférés lezárása vagy megnyitása

A 92-es függvényt használjuk egy állomány bizonyos részének lezárására úgy, hogy az megosztható legyen számos program között, miközben egyik program sem zavarja a másik működését. Ha egy program az állomány egy részét lezárja, akor használhatja vagy megváltoztathatja az állománynak ezt a részét mindaddig, míg az lezárt. Biztosak lehetünk abban, hogy amíg ez a rész lezárva marad, más program nem tudja azt használni. Természetesen az állománylezárást csak az állománymegosztási műveletekkel kapcsolatban használjuk.

Hat paraméter határozza meg, hogy egy állomány melyik részét zárjuk le. Az *AL* mutatja, hogy lezárjuk ($AL=0$) vagy megnyitjuk ($AL=1$) az állomány egy részét. A *BX* adja meg az

állományfoglaltsági jelzőt. A *CX*-et és a *DX*-et együtt kezeljük, egy 4 bájt hosszú integerként, amely meghatározza a lezárt résznek az állományban levő bájteltolását. Az *SI* és a *DI* is egy 4 bájt hosszú integert alkot, ez határozza meg a zárt rész hosszát. Ezekben a regiszterpárokban az első regiszter (a *CX* vagy az *SX*) adja meg az egész szám magas értékű részét.

Számunkra nem megengedett, hogy az állományrészeket darabonként vagy kombinációban nyissuk meg: a megnyitási kérésnek pontosan meg kell egyeznie az előzetesen lezáró kéréssel. Vigyázzunk, hogy mielőtt lezárnánk egy állományt, szüntessük meg részeinek lezárását; mert az elvárhatókkal ellentétben, egy állomány lezárása nem takarítja el szükségszerűen az állományunkban maradó részlezárásokat.

A 93-as függvény (hexa 62): a PSP-cím lekérdezése

A 98-as függvény kérdezi le a program szegmenselőtag címét, és adja azt vissza szegmensbekezdés címként a *BX*-ben. A DOS konvencionális világában a programok a PSP-jüket a kódszegmens első 256 bájtjába helyezik. Ez azt jelenti, hogy a PSP bekezdéscíme ugyanaz, mint a kódszegmens-regiszter (*CS*) tartalma. A személyi számítógépek és a DOS komplexebbé válásával azonban nem biztos, hogy mindig ilyen egyszerű lesz. Pl. az AT-ben levő 80286-os mikroprocesszor védett üzemmódjában a szegmensregisztereket különleges módon kezeljük. Ez a szolgáltatás remélhetően azért van, hogy egy állandó és megbízható módot adjon a szegmensregiszterek jövőbeli kezeléséhez.

18 A DOS-szolgáltatások összefoglalása

Bevezetés • 296

Rövid összefoglalás • 296

Részletesebb összefoglalás • 299

Bevezetés

Ez a fejezet a DOS szolgáltatásrutinjainak összegzése, és az a célja, hogy gyors referenciakönyvként használjuk. (☛ Az egyes szolgáltatások specifikus működésének részleteiről a megjegyzéseket l. a 15–17. fejezetekben.) Ha megértettük a DOS-szolgáltatásokat, ezek a táblázatok az összes szükséges programozási ismeretet biztosítják majd.

Rövid összefoglalás

Kilenc DOS-megszakítást a megszakításszámukkal hívunk. A 18.1. táblázatban ezekből öt megszakítást sorolunk fel. A táblázatban be nem mutatott négy megszakítás különleges célokra használható: a 33-as (hexa 21) függvényhívási megszakítás, a 80 DOS függvény egyikének meghívására használjuk; a 34–36-os megszakítások címmegszakítások, velük a speciális szubrutinokra mutathatunk. (☛ A továbbiakat l. a 15. fejezetben.)

A következő oldalon a 18.2. táblázatban felsorolt általános DOS-függvényeket a 33-as megszakítással (hexa 21) hívjuk meg; a függvényszámot az *AH*-regiszterbe helyezzük. Az általános függvényeket bármely DOS-változattal használhatjuk. (☛ Részletesebben l. a 16. fejezetben.)

Az új kiterjesztett DOS függvényhívásokat csak a DOS 2.00-val vagy annál későbbi változatokkal lehet használni. A 33-as megszakításon (hexa 21) keresztül hívjuk ezeket, és a függvényszámot az *AH*-regiszterbe helyezzük. (☛ A továbbiakat l. a 17. fejezetben.) A 18.3. táblázat sorolja fel az összes új DOS-függvényt, beleértve azokat, amelyeket a DOS 3.00-val vezettek be. (Ezeket a függvényeket nem lehet használni a korábbi változatokkal.)

18.1. táblázat
Az öt fő DOS-megszakítás

Megszakítás		Megnevezés
Dec	Hexa	
32	20	A program befejezése: rendes befejezés
37	25	Abszolút lemezolvasás
38	26	Abszolút lemezírás
39	27	A program befejezése a memóriában marad
47	2F	Nyomtatóspool vezérlés (DOS 3.00 és az ezt követő változatok)

18.2. táblázat
Az általános DOS-függvények

Függvények		Megnevezés
Dec	Hexa	
0	0	Befejezés: a program befejezése
1	1	Billentyűzetbemenet (echóval)
2	2	Monitorkimenet
3	3	Soros bemenet
4	4	Soros kimenet
5	5	Nyomtatókimenet
6	6	Közvetlen billentyűzet és monitor I/O
7	7	Közvetlen billentyűzetbemenet echó nélkül
8	8	Billentyűzetbemenet echó nélkül
9	9	Stringkiírás
10	A	Tárolt billentyűzetbemenet
11	B	A billentyűzetbemenet állapotának ellenőrzése
12	C	„Töröld a billentyűzetet és hajtsd végre”
13	D	A lemez alaphelyzetbe állítása
14	E	Az aktuális lemezegység kiválasztása
15	F	Állománymegnyitás
16	10	Állománylezárás
17	11	Az első illeszkedő állomány keresése
18	12	A következő illeszkedő állomány keresése
19	13	Állománytörlés
20	14	Soros állományrekord olvasása
21	15	Soros állományrekord írása
22	16	Állománylétrehozás
23	17	Állományátnevezés
25	19	Aktuális lemezegység jelentése
26	20	Lemez transzferterület beállítás
27	1B	FAT-információ és az aktuális lemezegység lekérdezése
28	1C	FAT-információ, bármely lemezegység lekérdezése
33	21	Random állományrekord olvasása
34	22	Random állományrekord írása
35	23	Az állományméret lekérdezése
36	24	A random rekordmező beállítása
37	25	A megszakításvektor beállítása
38	26	Programszegmens létrehozása
39	27	Random állományrekordok olvasása
40	28	Random állományrekordok írása
41	29	Az állománynév elemzése
42	2A	A dátum lekérdezése
43	2B	A dátum beállítása
44	2C	Az idő lekérdezése
45	2D	Az idő beállítása
46	2E	A lemezre írás ellenőrzése

18.3. táblázat

A DOS 2.00 és a későbbi változatok új DOS-függvényei

Függvény		Megnevezés
Dec	Hexa	
47	2F	A DTA-cím lekérdezése
48	30	A DOS-változatszám lekérdezése
49	31	**KEEP**: a „program befejezése memóriában maradással" függvény fejlettebb változata
51	33	A megszakításbillentyű *Ctrl-Break* lekérdezése és beállítása
53	35	A megszakításvektor lekérdezése
54	36	A lemez szabad területének lekérdezése
56	38	Az országfüggő-információ lekérdezése
57	39	Tárgymutató létrehozása
58	3A	Tárgymutató törlése
59	3B	Aktuális tárgymutató változtatása
60	3C	Állománylétrehozás
61	3D	Állománymegnyitás
62	3E	Állományfoglaltság jelzőjének lezárása
63	3F	Olvasás állományból vagy készülékről
64	40	Írás állományba vagy készülékre
65	41	Állománytörlés
66	42	Az állománymutató mozgatása
67	43	Az állományattribútumok lekérdezése vagy beállítása
68	44	**IOCTL**: I/O berendezések vezérlése
69	45	**DUP**: állományfoglaltság másolása
70	46	**CDUP**: foglaltságjelző másolásának kikényszerítése
71	47	Az aktuális tárgymutató lekérdezése
72	48	Memóriafoglalás
73	49	A lefoglalt memória felszabadítása
74	4A	**SETBLOCK**: a lefoglalt memóriatömb méretének módosítása
75	4B	Program betöltése és végrehajtása
76	4C	A folyamat befejezése
77	4D	Az alprogram visszaadott kódjának lekérdezése
78	4E	**FIND FIRST**: állománykeresés indítása
79	4F	**FIND NEXT**: állománykeresés folytatása
84	54	Az ellenőrző állapot lekérdezése
86	56	Állományátnevezés
87	57	Az állomány dátumának és idejének lekérdezése vagy beállítása
		DOS 3.00 függvények
89	59	A kiterjesztett hibakód lekérdezése
90	5A	Időleges állomány létrehozása
91	5B	Új állomány létrehozása
92	5C	Állomány-hozzáférés lezárása vagy megnyitása
98	62	PSP-cím lekérdezése

Részletesebb összefoglalás

Az előbbi részben röviden felsoroltuk az összes DOS-szolgáltatást, az egyes szolgáltatásokat a függvényszámuk révén lehetett megtalálni. Ebben a részben kiterjesztettük a felsorolást, bemutatjuk a bemeneti és kimeneti paraméterek regiszterbeállításait.

Mivel minden egyes új DOS-változat új függvényeket vezet be, amelyeket a korábbi változatokkal nem lehet használni, ebben a táblázatban szerepelnek a DOS-változatok számai a következők szerint kódolva:

DOS 1: A függvények az összes DOS-változattal használhatók.
DOS 2: A függvényeket a DOS 2.00-val és a későbbi változatokkal lehet használni.
DOS 3: A függvényeket a DOS 3.00-val és a későbbi változatokkal lehet használni.

18.4. táblázat
A DOS-szolgáltatások összefoglalása

Szolgáltatás	Függ-vény, hexa	Regiszter input	Regiszter output	Változat
Programvezérlő függvények				
Befejezés: a program vége	0	*AH* = 00		DOS 1
Programszegmens alkotása	26	*AH* = 26 *DX* = szegmensszám		DOS 1
KEEP: fejlett "program befejezés memóriában maradva"	31	*AH* = 31 *AL* = visszaadott kód *DX* = a szabad memóriaszegmens címe	*SX* = visszaadott kód	DOS 2
Vezérelt megszakítás *(Ctrl-Break)* lekérdezése és beállítása	33	*AH* = 33 *AL* = 00 lekérdezni *AL* = 01 beállítani	*AX* = visszaadott kód *DL* = aktuális állapot 00 = ki; 01 = be	DOS 2
EXEC: program töltése és végrehajtása	48	*AH* = 4B *AL* = alfüggvénykód *DS* : *DX* = az ASCIIZ-karakterlánc mutatója *ES* : *BX* = a vezérlőtömb mutatója	*AX* = visszaadott kód	DOS 2
Befejező eljárás	4C	*AH* = 4C *AL* = visszaadott kód		DOS 2
Az alprogram visszaadott kódjának lekérdezése	4D	*AH* = 4D	*AX* = visszaadott kód	DOS 2
A PSP-cím lekérdezése	62	*AH* = 62	*BX* = A PSP szegmenscíme	DOS 3

Szolgáltatás	Függ-vény, hexa	Regiszter input	Regiszter output	Változat
Billentyűzetvezérlő függvények				
Billentyűzet-bemenet visszhanggal	1	*AH* = 01	*AL* = bevitt karakter	DOS 1
Közvetlen billentyűzet-bemenet visszhang nélkül	7	*AH* = 07	*AL* = bevitt karakter	DOS 1
Billentyűzet-bemenet visszhang nélkül	8	*AH* = 08	*AL* = bevitt karakter	DOS 1
Tárolt billentyűzet bemenet	A	*AH* = 0A *DS*:*DX* = mutató a bemeneti tárra		DOS 1
A billentyűzet-bemeneti állapot ellenőrzése	B	*AH* = 0B	*AL* = FF, ha van karakter *AL* = 00, ha nincs karakter	DOS 1
A billentyűzet törlése és a függvény elvégzése	C	*AH* = 0C *AL* = függvény száma (1, 6, 7, 8 vagy A) *DL* = ha kimeneti karakter	*AL* = ha bemeneti karakter	DOS 1
Képernyővezérlő függvények				
Képernyőkimenet	2	*AH* = 02 *DL* = kimeneti karakter		DOS 1
Képernyőkarakterek	9	*AH* = 09 *DS*:*DX* = a kimeneti karakterek mutatója		DOS 1
Konzol I/O függvények				
Közvetlen billentyűzet vagy monitor I/O karakter	6	*AH* = 06 *DL* = kimeneti karakter ha FF, bemeneti karakter, ha <FF	*AL* = bevitt karakter	DOS 1
Egyéb I/O függvények				
Soros bemenet	3	*AH* = 03	*AL* = bevitt karakter	DOS 1
Soros kimenet	4	*AH* = 04 *DL* = kimeneti karakter		DOS 1
Nyomtatókimenet	5	*AH* = 05 *DL* = kimeneti karakter		DOS 1

Szolgáltatás	Függ-vény, hexa	Regiszter		Változat
		input	output	
Lemezkezelés függvényei				
A lemez visszaállítása	D	*AH* = 0D		DOS 1
Az aktuális lemezegység megválasztása	E	*AH* = 0E *DL* = lemezegység-azonosító	*AL* = lemezegység-számláló	DOS 1
Az aktuális lemezegység jelentése	9	*AH* = 19	*AL* = az alaplemezegység kódja	DOS 1
A lemezátviteli terület beállítása	1A	*AH* = 1A *DS* : *DX* = mutató a DTA-ra		DOS 1
A FAT-információ, az aktuális lemezegység lekérdezése	1B	*AH* = 1B	*AL* = a szektorok száma az allokációs egységben *CX* = bájt/szektor *DX* = az allokációs egységek száma *DS* : *DX* = mutató a FAT-azonosító bájtra	DOS 1
A FAT-információ, bármely lemez-egység lekérdezése	1C	*AL* = 1C *DL* = lemezegység-azonosító	*CX* = bájt/szektor *AL* = a szektorok száma az allokációs egységben *DX* = az allokációs egységek száma *DS* : *DX* = mutató a FAT azonosító bájtja	DOS 1
A lemezre írás ellenőrzésének beállítása	2E	*AH* = 2E *AL* = ellenőrző-kapcsoló 00 = ki 01 = be *DL* = 00		DOS 1
A random rekordmező beállítása	24	*AH* = 24 *DS* : *DX* = mutató az FCB-re		DOS 1
A random állomány rekordjainak olvasása	27	*AH* = 27 *CX* = rekordszám *DS* : *DX* = mutató az FCB-re	*AL* = visszaadott kód *CX* = tényleges rekordszám	DOS 1
A random állomány rekordjainak írása	28	*AH* = 28 *CX* = rekordszám *DS* : *DX* = mutató az FCB-re	*CX* = tényleges rekordszám	DOS 1

Szolgáltatás	Függvény, hexa	Regiszter		Változat
		input	output	
Az állománynév elemzése	29	*AH* = 29 *DS* : *SI* = mutató az utasítás sorra *ES* : *DI* = mutató az FCB-re *AL* = elemzésvezérlő-bitek	*DS* : *SI* = az utasítás következő sorára mutat *AL* = visszaadott kód *ES* : *DI* = mutató az FCB-re	DOS 1
CREAT: állomány létrehozása	3C	*AH* = 3C *CX* = állomány-attribútum *DS* : *DX* = az ASCIIZ-karakter-sorozat mutatója	*AX* = állomány-foglaltság jelző vagy a visszaadott kód	DOS 2
Állománynyitás	3D	*AH* = 3D *AL* = hozzáférési kód (l. 17.8. ábra) *DS* : *DX* = az ASCIIZ-karaktersorozat mutatója	*AX* = állományfoglaltság jelző vagy a visszaadott kód	DOS 2
Az állomány-fogó zárása	3E	*AH* = 3E *BX* = állományfoglaltság	*AX* = visszaadott kód	DOS 2
Olvasás állományról vagy készülékről	3F	*AH* = 3F *BX* = állomány-foglaltság *CX* = olvasandó bájtok *DS* : *DX* = mutató a DTA-tárra	*AX* = az olvasott bájtok száma vagy a visszaadott kód	DOS 2
Írás az állományra vagy készülékre	40	*AH* = 40 *BX* = állományfogó *CX* = írandó bájtok *DS* : *DX* = mutató a DTA-tárra	*AX* = az írt bájtok száma vagy a visszaadott kód	DOS 2
Állománytörlés	41	*AH* = 41 *DS* : *DX* = az ASCIIZ-karaktersorozat mutatója	*AX* = visszaadott kód	DOS 2
Az állomány-mutató mozgatása	42	*AH* = 42 *AL* = eljáráskód *CX* : *DX* = eltolásérték	*AX* = visszaadott kód, ha van *CF* *DS* : *AX* = az új mutató elhelyezkedése, ha *CF* nincs beállítva	DOS 2

Szolgáltatás	Függvény, hexa	Regiszter		Változat
		input	output	
CHMOD: az állomány- attribútumok lekérdezése vagy beállítása	43	AH = 43 AL = a lekérdezés vagy a beállítás kódja 00 = kérdezze le a CX-ből 01 = állítsa be a CX szerint CX = attribútum, ha a beállítás van használatban DS : DX = az ASCIIZ-karakter mutatója	AX = visszaadott kód CX = lekérdezéskor attribútum	DOS 2
DUP: az állomány- foglaltság duplikálása	45	AH = 45 AH = 45 BX = állomány-foglaltság	AX = állomány-foglaltság vagy visszaadott kód	DOS 2
CDUP: az állomány fejlécének duplikálása	46	AH = 46 BX = meglevő állományfoglaltság CX = második állományfoglaltság	AX = visszaadott kód CX = állományfoglaltság	DOS 2
FIND FIRST: az állomány- keresés indítása	4E	AH = 4E CX = attribútum, amely szerint keresünk DS : DX = az ASCIIZ-karakter mutatója	AX = visszaadott kód	DOS 2
Az állomány- keresés folytatása	4F	AH = 4F DS : DX = információ a **FIND FIRST** hívásból	AX = visszaadott kód	DOS 2
Az állomány újraelnevezése	56	AH = 56 DS : DX = az ASCIIZ-karakter mutatója (régi név) ES : DI = az ASCIIZ-karakter mutatója (új név)	AX = szabványosan kiegészített visszaadott kód	DOS 2
A kiterjesztett hibakód lekérdezése	59	AH = 59 BX = 0000	AX = hibakód BH = hibaosztály BL = tevékenység CH = hely	DOS 3
Időleges állomány alkotása	5A	AH = 5A DS : DX = mutató a tárgymutató útvonalnévre CX = állományattribútum	AX = hibakód, ha van CF DS : DX = útvonalnév, ha nincs CF	DOS 3

Szolgáltatás	Függvény, hexa	Regiszter input	Regiszter output	Változat
Új állomány alkotása	5B	AH = 5B $DS : DX$ = mutató az útvonalnévre CX = állományattribútum	AX = foglaltság vagy visszaadott kód AX = foglaltság vagy visszaadott kód	DOS 3
Az állomány-hozzáférés zárása vagy nyitása	5C	AH = 5C AL = 0 (zár) vagy 1 (nyit) BX = állományfoglaltság $CX : DX$ = eltolás záráshoz $SI : DI$ = zárandó összeg	AX = hibakód	DOS 3
Tárgymutatófüggvények				
MKDIR: tárgymutató alkotása	39	AH = 39 $DS : DX$ = az ASCIIZ-karakter mutatója	AX = visszaadott kód	DOS 2
RMDIR tárgymutató eltávolítása	3A	AH = 3A $DS : DX$ = az ASCIIZ-karakter mutatója	AX = visszaadott kód	DOS 2
CHDIR: az aktuális tárgymutató változtatása	3B	AH = 3B $DS : DX$ = az ASCIIZ-karakter mutatója	AX = visszaadott kód	DOS 2
Az aktuális tárgymutató lekérdezése	47	AH = 47 DL = lemezegység-azonosító $DS : SI$ = mutató az adatterületre	AX = visszaadott kód, ha van CF $DS : SI$ = útvonalnév, ha nincs CF	DOS 2
Dátum- és időfüggvények				
Dátumlekérdezés	2A	AH = 2A	AL = a hét napjai; 0 = Vasárnap; 6 = Szombat CX = év (1980 – 2099) DH = hónap DL = nap	DOS 1
Dátumbeállítás	2B	AH = 2B CX = év (1980 – 2099) DH = hónap DL = nap	AL = visszaadott kód	DOS 1
Időlekérdezés	2C	AH = 2C	CL percek CH = órák DL = század-másodpercek DL = másodpercek	DOS 1

Szolgáltatás	Függ-vény, hexa	Regiszter input	Regiszter output	Változat
Időbeállítás	2D	*AH* = 2D *CL* = percek *CH* = órák *DL* = századmásodpercek *DH* = másodpercek	*AL* visszaadott kód	DOS 1
Az állomány dátumának és időpontjának lekérdezése vagy beállítása	57	*AH* = 57 *AL* = a lekérdezés vagy a beállítás kódja 00 = lekérdezés; 01 = beállítás *BX* = állományfoglaltság *CX* = idő, ha *AL*=01 *DX* = dátum, ha *AL* = 01	*AX* = szabványos kiterjesztett kód *CX* = idő, ha *AL* = 00 *DX* = dátum, ha *AL* = 00	DOS 2
Egyéb függvények				
A megszakítás-vektor beállítása	25	*AH* = 25 *AL* = megszakításszám *DS* : *DX* = mutató a megszakítás-kezelőre		DOS 1
A DOS-változat számának lekérdezése	30	*AH* = 30	*AL* = főváltozatszám *AH* = alváltozatszám *BX* = 0000 *CX* = 0000	DOS 1
A megszakítás-vektor lekérdezése	35	*AH* = 35 *AL* = megszakításszám	*AX* = visszaadott kód *ES* : *BX* = megszakításvektor	DOS 2
Országfüggő információk lekérdezése	38	*AH* = 38 *DS* : *DX* = mutató a 32 bájtos tárra *AL* = 00, ha szabványinformáció, vagy országkód, vagy FF; ha országkód >255 *BX* = országkód, ha *AL* = *FF*	*AX* = visszaadott kód, ha van *CF* *DS* : *DX* = információ *BX* = országkód	DOS 2
IOCTL: I/O vezérlő készülékekhez	44	*AH* = 44 *AL* = az alfüggvény kódja *BL* = lemezegységszám *BX* = állományfoglaltság *CX* = olvasandó vagy írandó bájtok száma	*AX* = visszaadott kód, ha van *CF* *AX* = az olvasott vagy írt bájtok száma, ha nincs *CF* *DX* = vezérlő adatbitek	DOS 2

Szolgáltatás	Függ-vény, hexa	Regiszter input	Regiszter output	Változat
Memóriafüggvények				
A memória kiutalása	48	*AH* = 48 *BX* = a fejezetben igényelt memória	*AX* = szegmenscím vagy visszaadott kód, ha van *CF* *BX* = a legnagyobb rendelkezésre álló tömbméret	DOS 2
A kiutalt memória felszabadítása	49	*AH* = 49 *ES* = visszaadandó szegmenstömb	*AX* = visszaadott kód	DOS 2
SETBLOCK: A kiutalt memóriatömb módosítása	4A	*AH* = 4A *BX* = a fejezetben igényelt méret *ES* = a tömb szegmenscíme	*AX* = visszaadott kód *BX* = maximális méret	DOS 2

19 Programépítés

Bevezetés • 308

Programinterfészek • 308

Programmodulok összeállítása • 310

Az 1. lépés: a forráskód megírása • 310
A 2. lépés: a forráskód lefordítása • 310
A 3. lépés: programok szerkesztése • 311
A 4. lépés: állományformátumok konvertálása • 311
Az 5. lépés: tárgykódú könyvtárak létrehozása • 312

A DOS LINK-programjának használata • 313

Egyedi program szerkesztése • 313
Egy program összeszerkesztése a fordító könyvtárával • 314
Programok összeállítása • 314

Bevezetés

Már többször megállapítottuk, hogy a PC-család programozásának legbölcsebb megközelítése az, ha csaknem az összes programunkat magasszintű programnyelven írjuk (pl. BASIC, Pascal vagy C nyelven) és a DOS-vagy a BIOS-szolgáltatásokat csak arra használjuk, amire a magasszintű programnyelv nem alkalmas. Alkalmanként a különleges feladatok végrehajtására saját assembly nyelvű rutinokat is alkothatunk, amelyek a programozási nyelvből vagy a rendszer szolgáitatásaiból egyébként nem állnának rendelkezésünkre. Amikor egyetlen programozási nyelven írunk programokat, valójában a többi nyelv ismeretére nincs is szükségünk, elegendő a felhasználandó nyelv kézikönyvének ismerete. Ha mégis szükségünk van arra, hogy kitörjünk egy nyelv korlátai közül, és hozzáférjünk a rendszerrutinok némelyikéhez, ill. összekapcsoljuk egy programmá azt, amit különböző nyelvekben írtunk: akkor jobban bele kell mélyednünk mind DOS, mind a programozási nyelvek sajátosságaiba. A DOS-ba azért, hogy megtudjuk, miként kapcsolódnak a programok össze; a programozási nyelvbe pedig azért, hogy kitaláljuk a programinterfészekre vonatkozó előírásokat. (Ezek teszik lehetővé, hogy a különböző nyelvek egymással kommunikáljanak.)

Ez a fejezet néhány általános meggondolást mutat be, amely a legtöbb programozási nyelv fejlett felhasználására alkalmazható; azaz programinterfészek építésére és programok összekapcsolására a DOS LINK segédprogramjával. A következő fejezet a programozási nyelvet és az őket életre keltő nyelvi fordítókat tárgyalja. A 20. fejezetben majd ezeknek a nyelveknek néhány műszaki jellemzőjére is rámutatunk, amelyeket figyelembe kell vennünk, valahányszor assembly nyelvi rutinokhoz kapcsoljuk őket.

Programinterfészek

Egy programinterfész egy assembly nyelvű réteg, amely a magasszintű programnyelvben írt program számára lehetővé teszi, hogy egy assembly nyelvű szubrutinnal kommunikáljon. A programinterfésznek két fő része van: a *vezérlőinterfész* és az *adatinterfész*.

A **vezérlőinterfész** kezeli a hívás és a visszaadás folyamatát, azaz kezeli a számítógép vezérlésének hibamentes átadását az egyik modultól a másikhoz és onnan vissza.

A vezérlőinterfész egyébként trükkös program is lehet. Rendkívül egyszerű, ha helyesen használjuk, de hihetetlen bonyodalmakat okozhat, ha a legkisebb részletet is elhibázzuk.

Az **adatinterfész** lehetővé teszi az összekapcsolt két oldal érintkezését, és a közös adatok pontos megértését. Ehhez tudni kell, hogy a két oldal hogyan találja meg és dolgozza fel az adatot, és milyen az adatok formátuma. Biztosítani kell, hogy a két oldal azonosan értelmezze az interfészt. (☛ Erről részletesebben a következő fejezetben szólunk.)

A programinterfész tervezése csak egy része a programkapcsolási eljárásnak. Mind a három programelemnek – a programhívásnak, a hívott rutinnak és az interfésznek – a következőt kell elvégezni a sikeres együttműködésért:

A programnak meg kell találnia útját a szubrutinhoz. A szabvány PC-család 8088 bázisú rendszerében egy alprogramot kétféle módot lehet hívni: megszakításon (interrupton) vagy CALL-utasításon keresztül. Már tudjuk, hogy a DOS- és BIOS-szolgáltatásokat az INT-utasítás használatával, megszakításokon keresztül hívjuk, és a szolgáltatásrutin címe a megszakítás számában szerepel. A legtöbb assembly nyelvű programot és szubrutint a programozási nyelvünkből CALL-utasítással hívunk. A címek a program- vagy szubrutinnevekkel társulnak, amelyeket a szerkesztési eljárás (linking) állapít meg. (☛ A további részleteket l. az összeszerkesztésnél).

Kétféle **CALL**-utasítás van: a **NEAR CALL** és a **FAR CALL**. A **NEAR CALL** a 64K-s aktuális kódszegmensen belül keresi meg az alprogramot, és nem igényli a *CS*-regiszter változtatását. Ezzel ellentétben a **FAR CALL** az aktuális *CS*-en kívüli szubrutint keres a **CALL**-utasításban levő teljes szegmentált cím használatával (amely megváltoztatja a *CS* beállítását). Néhány nyelv mind a két utasítást használja, néhány pedig csak az egyiket.

A szubrutinnak tudnia kell, mit csináljon, amikor véget ér. Egy szubrutin a leggyakrabban a hívóprogramhoz tér vissza, egy **NEAR**- vagy egy **FAR RET**-utasítással, de vannak más célok is. Kívánhatjuk pl. azt is, hogy a programot befejezzük, és a DOS-hoz térjünk vissza a szubrutinból. A szubrutin **RET**-utasítása nemcsak a hívóhoz tér vissza, hanem a vermet is kitakarítja. (Ezt hamarosan látni fogjuk).

A szubrutinnak tudnia kell, hogy milyen támogatást kap a hívótól. Ez a támogatókeret tartalmazza azt, hogyan vannak beállítva a regiszterek, és vajon van-e egy használható verem. Általánosságban a szegmensregiszterek olyanok, amilyeneknek lenniük kell: a *CS* a megfelelő kódszegmensre; a *DS* a hívóprogram adatainak helyére; az *SS* és az *SP* pedig a hívó vermére mutat.

A hívott alprogram általában folytathatja a hívó vermének használatát, de azt gyakorlatilag nem tudhatjuk meg, mennyi munkaterülettel rendelkezünk. Ha igényeink átlagosak – mondjuk 64 bájtnál kevesebbek – a hívó vermének elegendő. Ha ennél többre van szükség, az alprogramnak be kell állítania saját adatterületét a memóriában.

Ha a program információt (paramétert) ad át a szubrutinnak, akkor a programnak és a szubrutinnak is tudnia kell, hány paraméter van, hová kell azokat helyezni, ill. hogy azokat meg kell-e változtatni, vagy vissza kell-e adni. Leggyakrabban a programok és a szubrutinok rögzített számú paraméterekkel dolgoznak, de néhány nyelv változó számú paramétert is kezelhet. A paramétereket mindig a vermen keresztül adják át: közvetlenül, vagy közvetve. A közvetlen eljárás a vermen keresztül a paraméterek aktuális értékét adja át, a közvetett eljárás az értékek címét. Mindkét esetben a hívóprogramnak feltétlenül ismernie kell az alkalmazott módszert.

A nyelvtől függ, hogy melyik paraméterátadási eljárást használjuk; néhány nyelv nem tud értéket helyezni a verembe, csak címeket. Azok a nyelvek, amelyek címeket és értékeket is kezelhetnek, sokkal nagyobb szabadságot adnak annak eldöntésére, melyik eljárást használjuk. Az alkalmazott eljárás meghatározza, hogyan adjuk át a paramétereket az egyik programból a másikba. Ha pl. védeni akarjuk a hívó pramamétereit a hívott szubrutin által okozott változtatásoktól, akkor az eredeti értékeket beírjuk a verembe, és megőrizzük az eredeti értékek egy másolatát. De ha azt akarjuk, hogy a hívó paramétereit a hívott szubrutin megváltoztathassa, akkor az eredeti érték címét kell küldenünk a vermen keresztül.

A paraméterátadás az interfész legösszetettebb része, amely még tovább bonyolódik azokkal a különböző módozatokkal, ahogy a programnyelvek az adatokat és a vereminformációt kezelik. Összetettsége és a két nyelv közötti eltérések miatt ezt a kérdést azonban csak a következő fejezetben, a nyelvi összehasonlítás részben beszéljük meg.

A szubrutinnak bizonyos információkat meg kell őriznie. Bár az igény különböző helyzetben változhat, de van néhány alapszabály; pl. milyen információkat kell megőrizni, mit lehet, ill. mit nem lehet tenni egy alprogram hívásakor. (☛ Néhány hasznos ötletet itt is láthatunk, a 3. fejezetben pedig a továbbiakat.)

A megszakításokat fel lehet függeszteni, bár általában ez nem szerencsés. (Kivéve arra a rövid időre, amíg a szegmensregiszterek változnak; azonban a visszatérés előtt újból vissza kell azokat kapcsolni. ☛ L. a 3. fejezetet.)

Ha bármely szegmensregiszter értékét módosítjuk a veremben meg kell őrizni az eredeti értéket. Egy másik megőrzendő fontos regiszter (a legtöbb esetben) az alapmutató-regiszter

(*BP*), mivel ezt gyakran használjuk a veremben levő paraméterek helyének nyomon követésére. Konvenció szerint a hívóprogram nem várja el, hogy a munkaregiszter értékei megőrződjenek, így az összes munkaregiszter – *AX, BX, CX, DX, DI* és *SI* – szabadon változtatható ugyanúgy, mint az összes jelzőbit (flag). A hívó vermét meg kell őrizni, bár ennek konkrét megvalósítása a kitakarítási folyamat része.

A vermet ki kell takarítani, miután a szubrutin befejeződött. Négy dolog keveredhet össze a veremben: *a paraméterek;* a CALL-utasításból való *visszatérés címe;* a CALL-utasítás előtt mentett *regiszterértékek;* ill. a szubrutinból bizonyos *munkatár.*

Három ezekből a hátrahagyott dolgokból nem jelent nehézséget: az alprogramoktól elvárjuk, hogy a saját munkatárukat a veremből eltávolítsák; a kimentett regisztereket POP-utasításokkal távolátjuk el; a visszatérési címeket pedig a RET-utasítás távolítja el. Általában a paraméterek bonyolítják a takarítási eljárást, mivel az eltávolítás módja a különböző nyelvekben eltér. Néhány nyelv a szubrutintól elvárja, hogy – a veremből eltávolítandó bájtok számát a RET-utasításban meghatározva – távolítsa el a paramétereket. Más nyelvek a hívótól várják, hogy eltávolítsa azokat. (☛ Felhívjuk majd a figyelmet ezekre a különbségekre a 20. fejezetben.)

Mindezeket a programtervezési elemeket észben tartva, lépjünk vissza egy kicsit, lássuk, hogyan működik az eljárás; a program vagy szubrutin megalkotásától a más programokkal, ill. szubrutinokkal való kombinálásig!

Programmodulok összeállítása

Itt tárgyaljuk a programrészek összerakására, ill. a programmodulok összeállítására vonatkozó általános szabályokat. A szabványos DOS programozási eljárást használjuk, amely vonatkozik a következő fejezetben példaként használt összes programozási nyelvekre (kivéve az értelmezős BASIC-et, amely úgy tűnik, mindig kivételes eset). Először tekintsük át azokat a lépéseket, amelyek egy működő program létrehozásával kapcsolatosak!

Az 1. lépés: a forráskód megírása

A programozási nyelvünk utasításainak és szintaxisának használatával először meg kell írnunk a programokat. A programnak ezt a formáját hívjuk forráskódnak. A szabványos DOS konvenciókat használó programozási nyelvek forráskódja ASCII szövegfájlformátumban kell legyen. (☛ L. a *C* mellékletet.) Az értelmezős BASIC általában nem használja az ASCII szövegfájlformátumot a forrásprogramok tárolására, de esetenként használhatja. (Ahhoz, hogy ASCII-szövegfájlokat hozzunk létre a BASIC-kel, a SAVE-utasítás A-opcióját kell használnunk.) A konvenció szerint a forráskódállományoknak olyan állománynév kiterjesztésük van, amely tükrözi a használt programozási nyelv nevét (pl. BAS vagy C).

A 2. lépés: a forráskód lefordítása

Utasításra a forrásállományok a nyelvi fordítókba kerülnek, amelyeket fordítónak (compiler), ill. az assembly nyelv esetén assemblernek hívunk. (A BASIC-értelmező ismét különleges eset, amelyet nem veszünk figyelembe itt.) A fordító és az assembler a forráskódot gépi nyelvi

utasításokra fordítja; de nem konvertálja azt olyan formátumba, amely kész a végrehajtásra. Ehelyett a fordítók és az assemblerek a kimenetüket egy ún. tárgykódformátumban helyezik el. A tárgykódformátumot úgy tervezték, hogy avval a különböző tárgymodulokat egy nagyobb programba lehessen összekapcsolni. A tárgykódú állományok megállapodás szerint OBJ-állománynév kiterjesztésűek.

A 3. lépés: programok szerkesztése

A következő alapvető lépés a programok összeszerkesztése. A szerkesztő (linker) vagy linkeditor program a DOS esetében két fő feladatot hajt végre: a különálló tárgyi modulokat összeállítja (igény szerint), és létrehozza a közöttük szükséges összes kapcsolatot; ill. a modulokat egy tárgykódú formátumból egy EXE-formátumban levő, betölthető programba konvertálja.

Fő szempontunk: miként kapcsolódnak ösze a programmodulok, hogy egy EXE-állományt alkossanak. Ezt később ismét felvetjük ebben a fejezetben, miután a program előkészítésére vonatkozó két további lépést megbeszéltünk.

Eddig három fő program-előkészítési lépést említettünk: a program forráskódjának előállítását; a fordító, ill. az assembler programmal a tárgykód előállítását; a szerkesztő programmal pedig egy betölthető program előállítását. A folyamatban két másik idevonatkozó lépés van; a LINK-program által létrehozott .EXE-formátum konvertálása .COM-formátumra, és a tárgykódú könyvtárak használata a modulok tárolására.

A 4. lépés: állományformátumok konvertálása

Azok a programok, amelyek a lemezen .EXE formátumban vannak, nem készek a futásra. Amikor a lemezről a DOS a programot a memóriába betölti, előkészíti még néhány művelettel a programot a végrehajtásra. Ezek a műveletek pl. közlik a program helyét a memóriában, kiszámolják annak méretét, beállítják a használható vermet. De ha egy program működési konvenciói elegendően egyszerűek, a program induló fázisai pedig elegendően takarékosak; ez a betöltési előkészület elvégezhető előre az állomány .COM állományformátumba konvertálásával. A .COM-állomány a program pontos képe, ahogyan az a számítógép memóriájában megjelenik.

Míg a DOS-nak egy .EXE formátumú program előkészítésére sok munkát kell elvégeznie, egy .COM formátumú programban csak két dolgot kell tennie: létrehozza a programszegmens előtagot (☛ l. a 15. fejezetet); és beállítja a szegmensregisztereket.

Egy .EXE-állomány .COM állományformátumba konvertálására az EXE2BIN nevű DOS-programot használjuk. Nem lehet az összes .EXE-programot .COM-formátumra konvertálni. A feltételeknek eleget tevőket konvertálhatjuk vagy hagyhatjuk .EXE-formátumban is. Mindkét forma funkcionálisan ugyanaz, de a .COM-formátum tömörebb és kicsit gyorsabban töltődik be.

Általában nem a mi választásunktól függ, hogy egy .EXE-állomány konvertálható-e vagy sem. Ez mindig a programozási nyelv által létrehozott konvencióktól függ. A legtöbb magasszintű programnyelvnek szüksége van arra a külön támogatásra, amit az .EXE-formátum nyújt, ezért az ilyen programok nem konvertálhatók. (Az .EXE támogató műveleteinek jegyzékét l. az IBM DOS Technical Reference kézikönyvében.) Olyan programok, amelyek tisztán assembly nyelven készültek, szabadon választhatnak a két formátum közül. A BASIC és a

Pascal nyelvű programokban (és számos más fordítós nyelvben is) az .EXE-formátum kötelező. A C programozási nyelvben (Lattice/Microsoft C fordítót használva) írt programokkal, amelyeket néha az assembly szintű nyelv és a magasszintű nyelvek közötti nyelvként emlegetnek, mindkét utat követhetjük.

Nagyon egyszerűen és biztonságosan kitalálhatjuk, hogy egy program konvertálható-e .EXE-formátumból .COM-formátumra: egyszerűen azzal, hogy megpróbáljuk. Ha sikerül, akkor konvertálható, ha viszont az EXE2BIN vagy LOCATE azt mondja, hogy ez nem végezhető el, akkor nem konvertálható.

Az 5. lépés: tárgykódú könyvtárak létrehozása

A legtöbb magasszintű programozási nyelv a programozási nyelv a programunk működését támogató előkészített szubrutinok tucatját használja. Természetesen ezek már lefordított, tárgykódban lévő szubrutinok. Emellett rendkívül kényelmetlen, hogy e tárgykódok tucatjai heverjenek össze-vissza a lemezeinken. Az is bonyolult feladat, ha magunk határozzuk meg, hogy melyeket kell összeszerkeszteni a saját programunkkal. A megoldást a tárgykódú könyvtárak jelentik, amelyek összegyűjtött tárgykódok. Megállapodás szerint a könyvtáraknak .LIB-állománynév kiterjesztésük van.

A legtöbb magasszintű programozási nyelvhez használatra kész, szabványos, támogató szubrutinkönyvtár tartozik. Egy fordítónak számos könyvtára lehet, amely a szabványos rutinok különböző változatait nyújtja, pl. tartozhatnak hozzájuk olyan lebegőpontos rutinok, amelyek vagy használják a 8087 matematikai koprocesszort, vagy nem.

A DOS LINK-programja képes egy könyvtár átkutatására, hogy megtalálja és felhasználja azokat a szubrutinokat, amelyekre egy program összeszerkesztéséhez szükség van.

A könyvtár nélkül szembe kellene néznünk azzal a bosszantó feladattal, hogy nekünk kellene közölnünk a linkerrel, melyik tárgyi állományokra van szükség. Ha egyet is kihagynánk, az összeszerkesztés meghiúsulna; ha pedig egyet is beszerkesztenénk, amelyre ténylegesen nem volt szükség, a programunk szükségtelenül megnőne. A könyvtár használata lehetővé teszi számunkra, hogy ezeket elkerüljük.

A tárgykódú könyvtárakat legtöbb esetben úgy használjuk a fordítók támogatására, hogy azok teljesen az irányításunkon kívül esnek. Nem hozunk létre, nem módosítunk könyvtárakat, és közvetlenül sem választjuk ki, hogy mit használunk egy könyvtárból. Ehelyett az állományokat vagy közvetve választjuk programnyelvünk jellemzőinek használatával, vagy közvetlenül a LIB-programmal.

🕮 MEGJEGYZÉS: A LIB-program nem része az összes DOS-változatnak, bár annak kellene lennie. Az IBM DOS-változatai az 1.00-tól a 3.1-ig nem foglalják magukba a LIB-et. A DOS néhány nem-IBM változatába beletartozik, része néhány (de nem minden) fordítónak vagy assemblernek. Feltétlenül szükséges előfeltétel, hogy mielőbb hozzájussunk a LIB egy munkamásolatához.

Ha van egy LIB-másolat, akkor három lényeges célra használhatjuk. Egyszerűen a meglevő könyvtárak tartalmának átnézésére (ez igen izgalmas kísérlet lehet); a meglevő könyvtárakban levő modulok szelektív cseréjére, ha változtatni, ill. javítani kívánjuk a programozási nyelvvel adott könyvtárat; vagy saját könyvtárak alkotására.

A LIB-re vonatkozó DOS kézikönyvekben a dokumentáció részletesen elmagyarázza annak használatát, de ízelítőül azért néhány példát is ajánlunk kipróbálásra. Új könyvtár (TESTLIB nevű) előállításához írjuk be az alábbi utasítást:

LIB TESTLIB;

Egy meglevő könyvtár tartalmának listázásához és a lista LPT1-re küldésére (vagy bármely állományra, vagy a képernyőre):

LIB TESTLIB, LPT1;

Az X.OBJ-modul könyvtárhoz adására:

LIB TESTLIB + X;

A meglevő modul új változattal való lecseréléséhez:

LIB TESTLIB – X + X;

A LIB későbbi változatainál a – X + X helyett – + X kell. Disassembler vagy egyéb külön felhasználásra való modul kivételéhez:

LIB TESTLIB*X;

Programjaink általában számos szubrutinból állnak össze. A LIB-program szolgáltatásaiból csak akkor származhat előnyünk, ha jól szervezzük meg ezeket a szubrutinokat. Ha a szubrutinok forráskódjait állítjuk inkább össze egy forrásállományba, akkor kevés szükség van a LIB-re. Ha viszont szívesebben szerkesztünk a tárgykódú rutinokat, akkor a LIB éppen az általunk igényelt munkát végzi el: összegyűjti és megszervezi tárgykódú állományainkat. A szerző egyik működési stílust sem részesíti előnyben; használta mindkettőt a PC programozása során, és mindkettőt praktikusnak és hatékonynak találta. Ez valójában az ízlésünktől függ, de a Pascalban való programozásnak is következménye lehet. (☛ L. a 20. fejezetet.)

A DOS LINK-programjának használata

Most visszatérünk a programmodulok összeállítására, ill. a LINK-program használatára. A LINK-program dokumentációja az IBM DOS Technical Reference kézikönyvben átfogóan elmagyarázza működését, beleértve összetett vezérlőkapcsolóit is. Itt a leggyakoribb és leghasznosabb műveleteket öszegezzük, különösen ott, ahol ezek a következő fejezetben tárgyalt programozási nyelvek velejárói.

A LINK-program négy paramétert vesz fel, amelyet a következőképpen írhatunk:

LINK 1, 2, 3, 4;

Az *első* paraméter (1) a tárgykódú modulok kifejtett listájára vonatkozik (PROG1 + PROG2 + PROG3); a *második* (2) a kész program neve; *harmadik* (3) arra vonatkozik, hogy a szerkesztőképernyő kimenetét hová kell küldeni (pl. a nyomtatóra vagy a képernyőre); és a *negyedik* (4) a könyvtárak kifejtett listájára vonatkozik, ha használjuk azokat (BASCOM + MYLIB).

Egyedi program szerkesztése

Most nézzünk néhány gyakorlati példát. Elsőnek vizsgáljunk meg egy önálló programot! (☛Pl. az előző fejezetben bemutatott BEEP-programot.) Ennek szerkesztésére írjuk be:

LINK BEEP;

Egy egyszerű program szerkesztése (mint ezé is) egy .EXE-állományt állít elő.

Egy program összeszerkesztése a fordító könyvtárával

A következőkben vizsgáljuk meg, mi a leggyakoribb szerkesztési feladat. Amikor összeállítunk egy programot egy magasszintű nyelvben (pl. fordítós BASIC), és szükségünk van arra, hogy összekapcsoljuk azt annak szabványos könyvtárával; ebben az egyszerű esetben mi nem használunk semmilyen interfészt vagy más általunk előállított modult. Proframunk neve X, és a fordító könyvtár neve BASCOM.LIB. A LINK-utasítást a következőképpen írnánk:

LINK X, ,BASCOM;

Általában egy fordító olyan tárgyi modult generál, amely a könyvtárban keres minden egyebet, amire szüksége van. A Lattice/Microsoft C esetében van egy szabványos indítórutin, amelyet előtagmodulnak hívnak, és amelyet a programunk elejére kell szerkeszteni. Ha az előtagmodult C-nek, a programunkat X-nek, a könyvtárat pedig MC-nek nevezzük, akkor a szerkesztésük így nézne ki:

LINK C+X, X, ,MC

Vegyük észre a két új tételt ebben a példában! Először kifejezetten kértük a szerkesztőt, hogy két tárgykódú állományt kapcsoljon össze, a C.OBJ-t és az X.OBJ-t (amely a program, amit összeállítottunk). Másodszor konkrét nevet adtunk a kész .EXE-programnak, X-nek nevezve azt. Ezt a nevet meg kell adni, különben a szerkesztő az első tárgykódú modul nevét (a C-t) fogja használni. Ha ez utóbbit nem akadályozzuk meg, akkor az összes programunk ugyanazt a nevet (a C-t) kapja egyedi név (ebben az esetben az X) helyett. A kész programnak bármely általunk kívánt nevet adhatunk, beleértve a tárgyi modulok nevéhez egyáltalán nem kapcsolódó neveket is. Ebben a példában a szokásos utat követtük, vagyis a kész programunkat ugyanúgy X.EXE-nek neveztük, mint az X.OBJ nevű tárgykódú állományt.

Programok összeállítása

Most már be tudjuk mutatni, hogyan állítjuk össze a szerkesztővel a programmodulokat. Először vizsgáljunk meg egy olyan helyzetet, amelyben egy saját tárgykódú könyvtárat használunk assembly nyelvű interfészeink vagy összeállított szubrutinjaink, esetleg mindkettő tárolására. Íme, hogyan szerkesztenénk egy ilyen könyvtárat Pascal nyelven. A programot X-nek nevezzük, a könyvtárat pedig OURLIB-nek:

LINK X, ,PASCAL+OURLIB

A következőkben vizsgáljunk meg egy olyan esetet, ahol nem alkottunk tárgykódú modulkönyvtárat. Ehelyett egyszerűen két tárgykódú állományt akarunk összeszerkeszteni. Egyet a magasszintű programnyelvi fordítónkból, egyet pedig az assembly nyelvű interfészekből. Így kapcsolhatunk össze ilyen programot (a példában a nyelv a Pascal, a programot X-nek nevezzük, és az assembly nyelvű interfészt INTFACE-nek.):

LINK X+INTFACE, ,PASCAL

Természetesen a programmodulok összeállításának végtelen sok változata van. Ezek az alapvető példák a legszükségesebb információkat mutatják. Felfegyverkezve ezekkel az információkkal, áttérhetünk a következő területre, és megtudhatunk valamit arról, hogyan működnek a programozási nyelvek.

20 A programozási nyelvek

Bevezetés • 316

A nyelvek sajátosságai • 316

Az assembly nyelv • 317

Logikai szerkezet • 318
Az interfészek konvenciói • 318
Assembly programok írása és összeszerkesztése • 319

Az értelmezős és a fordítós BASIC • 321

A BASIC adatformátumai • 321
Az értelmezős BASIC assembly interfésze • 325
A fordítós BASIC assembly interfésze • 327

A Pascal nyelv • 330

A Pascal adatformátuma • 330
A Pascal assembly interfésze • 334

A C nyelv • 336

A C nyelv adatformátumai • 336
A C nyelv assembly interfésze • 338

Zárómegjegyzés • 340

Bevezetés

Ebben a fejezetben röviden megbeszéljük a programmodulok építésének, és a DOS LINK-programmal való összekapcsolásának általános elveit. Kitérünk azokra a programozási nyelvekre is, amelyeket a programmodulok építésére használunk; különös tekintettel azokra a nyelvi szempontokra, amelyekkel foglalkoznunk kell, ha a modulokat assembly nyelvű szubrutinokhoz akarjuk kapcsolni. A fejezet címe azt sejteti, hogy általában programozási nyelveket tárgyalunk ez azonban nem így van. Nagyon helyénvaló, hogy bármely témát elvonatkoztatva tárgyaljunk, de ha valamit el akarunk végezni, el kell jutni a konkrétumokig. Ha számítógépes programokat akarunk írni, akkor konkrét programozási nyelvekkel kell dolgoznunk, és egy programozási nyelv jóval sajátosabb, mint ahogy azt sok emberrel elhitették. Valójában általános programnyelv nem is létezik. Programokat csak olyan programozási nyelvekre vonatkozó fordítóval (compilerrel) vagy értelmezővel (interpreterrel) írhatunk, amelyet egy meghatározott gépre terveztek. Bár akadémikus számítógép-szakértők az ellenkezőjét akarják elhitetni, a gyakorlati igazság az, hogy a programnyelvek általános definícióból hiányzik sok olyan lényeges jellemző, amikre valós programok írásakor van szükségünk. Így tehát amikor egy fordítót (compiler) vagy egy értelmezőt (interpreter) alkotnak egy meghatározott programnyelvre (a BASIC-re) azért, hogy egy meghatározott számítógépen fusson (pl. a PC-n); az alapnyelvet változtatják, kiegészítik, hogy azt nyújtsa, amire valóban szükség van. A változtatások gyakran jelentősek, és minden esetben egy olyan programnyelvet adnak, amely hasonlít – de ugyanakkor különbözik is – az összes többi, ugyanolyan nevű programozási nyelvre.

Ezzel a rövid érveléssel vezettük be azt az egyszerű megállapítást, hogy ez a fejezet nem fedi (nem is fedheti) az összes programozási nyelvet – amely most létezik, vagy a jövőben jön létre. Mivel minden egyes fordító valójában előállítja a saját programozási nyelvét, ennek a fejezetnek nem célja a fő nyelvek általános szempontjait tárgyalni. Mindet nem, csupán néhány nyelvet választunk ki, négy népszerű és reprezentáns nyelvet: az assembly nyelvet, a BASIC-et, a Pascal-t és a C-t. Azután e kategóriákon belül kiválasztottuk e nyelvek öt jellemző változatát, ill. implementációját: az IBM Macro Assemblyt, az IBM értelemzős BASIC-et, az IBM fordítós BASIC-et, az IBM Pascalt és a Lattice/Microsoft C-t.

A nyelvek választását a múlt és a jövő befolyásolta, tapasztalatunk szerint ezek a nyelvek terjedtek el a legszélesebb körben a PC-családban. Nem lenne bölcs olyan programnyelveket használni, amelyek vagy nem állítanak elő szerkeszthető tárgykódot (mint az „olcsó assembler" a CHASM, a Borland's Turbo Pascal és a Logitech's Modula–2), vagy olyan tárgyformátumot használnak, amely nem kompatibilis a DOS-szerkesztővel (pl. a Digital Research nyelvcsaládja és a Computer Innovation's C – 86).

A szerző igen konzervatív a szerkesztés és a tárgykód-kompatibilitás ügyében. A „de facto" szabványként bevezetett DOS-tól való eltéréssel nagy kockázatot vállalunk, és a számítógépek PC-családjának fejlődésével szembe találjuk magunkat e problémákkal.

A nyelvek sajátosságai

A kiválasztott öt programozási nyelv valójában egy-egy család. Mindegyiknek számos változata van, és a legtöbb esetben több forrásból állnak rendelkezésre. Szerencsére a változatok közötti különbségek elég kicsik, ezért nem kell őket különböző nyelvnek tekinteni, mint a BASIC-et és a Pascalt.

Az assembly nyelv. Az assembly nyelvnek tárgyalását a Microsoft által készített IBM 1.00 változatú Macro Assemblerre alapozzuk. A Microsofttól, az IBM-től és olyan más számítógép-

gyártóktól, akik a Microsoft BASIC assemblerének licencét megvásárolták, számos változat ismert. Ezek a változatok csak a legkifinomultabb elemeikben térnek el, ezek miatt most nem kell aggódnunk.

Az értelmezős (interpreteres) BASIC. A fejezetben tárgyalt BASIC ezernyi arcot és változatot ölt. Az IBM PC felhasználói előtt az itt tárgyalt változat egyszerűen BASIC vagy BASICA néven vagy további változatnevekkel (C1.10, J1.00 és A2.10) ismert. Az IBM világon kívül BASIC-ként, Microsoft BASIC-ként vagy GW-BASIC-ként ismerhetik. Most nem törődünk a különbségekkel: csak a közös elemekkel foglalkozunk.

A fordítós (kompilált) BASIC. A fordítós BASIC tárgyalásunknál az IBM BASIC fordító 1.00 változata vezet bennünket. Az elvek, amelyeket megtárgyalunk más Microsoft BASIC fordítókra, pl. a Business BASIC-re is vonatkoznak.

A Pascal. A Pascalra vonatkozó alapként az IBM 1.00 változatát használjuk. A részletek, amelyeket megbeszélünk, ugyanilyen jól használhatók az IBM 2.00 változatra, és számos más Microsoft-változatra.

A C nyelv. A C tárgyalásánál a Lattice cég által készített Lattice/Microsoft C compiler 1.04 változatát használjuk. Vannak más beszerezhető változatok a Microsofttól, a Lattice-tól és a Life-boat-tól, amelyek az általunk leírt jellemzők mindegyikét magukba foglalják. Egyébként ezt a fordítót ne keverjük össze a Microsoft C fordítójával (3.00 változat és afelettiek), amely azután jött létre, hogy a Lattice/Microsoft C változatot kiadták.

Az assembly nyelv

Az assembly nyelvű programoknak két alapvető típusuk van: az assembly nyelvű szubrutin, amelyet más magasszintű programnyelvben írt programok hívnak meg, és az önálló assembly nyelvű program. A szubrutinok nagyban függenek a hívóprogramtól, mivel biztosítani kell a hívóprogram szerkezetét és szolgálatait. Az önálló assembly nyelvű programoknak kizárólag saját maguk szerkezetét és kiszolgálását kell nyújtaniuk, viszont meg kell oldaniuk azokat az alapvető működési kérdéseket, amellyel egy önálló programnak is szembe kell nézni. Assembly szubrutinokat viszonylag könnyű szerkeszteni, míg az assembly programok igen összetettek lehetnek.

A szubrutinok hamarabb felkeltik azok figyelmét, akiknek magasszintű programnyelv és bizonyos (BIOS vagy DOS) szolgáltatások közötti interfészrutinok építését kell megoldaniuk. Az önálló programokat általában azok a programozók kedvelik, akiknek valami olyasmit kell elvégezniük, amelyet sem a hagyományos magasszintű programozási nyelvük, sem a rendszerszolgáltatások nem nyújtanak.

Az assembly nyelv rövid tárgyalásában bemutatunk néhány módszert, amely segít felismerni az assembly nyelvű alprogramok és a magasszintű programnyelvek közötti interfészek konvencióit. Áttekintjük egy önálló assembly programírás folyamatát is, de meg sem kíséreljük, hogy megtanítsuk az assembly nyelv használatát, mert rendkívül összetett feladat. Az assembly ismeretek hiányának pótlására a könnyen elérhető assembly nyelvű programozási források tanulmányozása jó módszer lehet. Használhatjuk a számítógépgyártók által publikált assembly programokat is – pl. a BIOS-listákat – az IBM Technical Reference kézikönyvében. Kitűnő lehetőséget ad a legtöbb fordító, amelyet utasíthatunk programunkból assembly-szerű lista előállítására. Ez hasznos annak megismerésére, miként kezel a fordító bizonyos kódolási problémákat (amelyet a programozó a magasszintű programnyelvben levő megfelelő utasítások kiválasztásával vezérelhet), és arra is, hogy megismerjük a fordító által használt interfész konvenciókat. Egy hasonló, de kevésbé hasznos tanulási mód egy létező programot a DOS

DEBUG programjával betölteni, és utána a DEBUG U (unassemble) utasítását használva a programot végigfürkészni. Az itt következő rész mutatja be, hogyan hajtsuk végre a programkönyvtárakon belüli fürkésző műveletet. Ezen eljárások mindegyike segíthet különböző programozási technikák és trükkök megismerésében. E könyv szerzője is ezeket a módszereket használta.

Logikai szerkezet

Egy assembly nyelvű szubrutin elemeinek a megértése könnyű, ha a megjelenési sorrendjükben vannak. A 8. fejezetben már részletesen elmagyaráztuk: a logikai szerkezetet az interfészrutint öt egymásba ágyazott részként írjuk le.

- 1. szint: Általános assembler overhead
 - 2. szint: Szubrutin assembler overhead
 - 3. szint: Bemeneti kód
 - 4. szint: Paraméteradatok lekérdezése a hívótól
 - 5. szint: A ROM-BIOS- vagy a DOS-szolgáltatás kiváltása
 - 4. szint: Az eredmények visszaadása a hívónak
 - 3. szint: Kimenet a kódokból
 - 2. szint: A szubrutin assembler overhead befejezése
- 1. szint: Az általános assembler overhead befejezése

Ez az alapszerkezet követhető a rendszerszolgáltatásokra vagy a hagyományos assembly nyelvű szubrutinokra írt legtöbb szubrutinnál, de a tényleges kódolás minden egyes programnyelvnél változik.

Assembly szubrutinok írásakor a szabványos eszköz a PC-családban a Microsoft Macro Assembler, amelyet MASM-nak nevezünk, és amely különböző változatokban számos forrásból rendelkezésre áll. Az IBM PC-t pl. az 1.00 változattal vezették be, amely egy kisassemblert (makro kapacitás nélkül), és egy makroassemblert is magában foglal. Bármelyik assemblerváltozat (kivéve a fejlett assembly nyelvű munkákat) megfelelő a céljainknak.

Az interfészek konvenciói

Ha az assembly nyelvet már megismertük, a következő lépés az assembly nyelvre és programra vonatkozó interfész szokások megismerése. Az assembly nyelvű interfésznek tudnia kell, hogyan férjen hozzá a hívóprogram által átadott paraméterekhez, hogyan értelmezze az adatformátumot, hogyan küldje vissza a paramétereket. Ha a nyelv dokumentációjában nincs elegendő ilyen jellegű információ, néhány egyszerű mód van arra, hogy magából a nyelvből kitaláljuk.

Adott assembly nyelvű szubrutinok belső részeit is megvizsgálhatjuk, ez a fordítótól származó listázások tanulmányozása révén szerzett ismeretektől némiképp eltérő perspektívát ad. Ez a technika nemcsak az assembly nyelvű rutinokra vonatkozó interfészkonvenciók részleteit adja, hanem sajátos programozási példákat is, amelyek modellként szolgálhatnak.

Először ki kell választanunk egy megfelelő szubrutint. A tanulmányozásra a leghozzáférhetőbb szubrutinok gyakran azok, amelyek a fordítót kísérő könyvtár részei. Általában könnyebb egy olyan fordítójellemzőt (pl. I/0, képernyőkezelés vagy aritmetika) választani, amely érdekel minket, és azután kitalálni, hogy ehhez a jellemzőhöz milyen szubrutinok tartoznak.

Ezt követően nézzük meg a könyvtármodulok neveit (mert nem biztos, hogy a modulokon belül levő szubrutinok neve is). Ezt a LIB-programkönyvtár tartalmának kilistázására vonatkozó ismereteink felhasználásával érhetjük el. (☛ A 19. fejezetben tárgyaltuk.) Tegyük fel, hogy létezik egy LANGF.LIB nevű könyvtár az állományban. A könyvtár kilistázást egy másik, LISTING-nek nevezett állományba a következő DOS-utasítással irányíthatjuk (irányíthatnánk ezt a képernyőre vagy a nyomtatóra is):

LIB LANG, LISTING;

A könyvtárat átnézve megtaláljuk azt a szubrutint, ami minket érdekel, és annak a modulnak a nevét, amely része annak – nevezzük XMOD-nak. Ezt követően kérjük a LIB-et, hogy válassza ki az XMOD-ot a könyvtárból, hogy azzal dolgozhassunk:

LIB LANG * XMOD;

A * műveleti jel közli a LIB-bel, hogy a modulnak egy másolatát készítse el külön tárgykódú állományként; ebben az esetben az állományt XMOD.OBJ-nak nevezi.

Ennél a pontnál belenézhetünk az XMOD.OBJ belsejébe, de ez az állomány nem odatartozó szerkesztési információkat tartalmaz, amely csak utunkban lenne. Ehelyett az XMOS.OBJ-hez két lépés felhasználásával tiszta gépi utasítások készletével fordulunk. Először egy .EXE programállománnyá konvertáljuk:

LINK XMOD;

Ez egy XMOD.EXE-nek nevezett állományt ad. (Hagyjunk figyelmen kívül minden hibaüzenetet!) Ezután, hogy megszabaduljunk az .EXE-állomány fölös részeitől, az XMOD.EXE-t egy .COM-állománnyá alakítjuk:

EXE2BIN XMOD.EXE XMOD.COM

Ennél a pontnál van egy XMOD.COM-nak nevezett állományunk, ami nem tartalmazhat mást, mint gépi kódú utasításokat az összes sallang nélkül. Most készen állunk arra, hogy használjuk a DEBUG-ot az utasítások olvasható assembly formátumba konvertálására. Először megjegyezzük az XMOD.COM méretét, utána betöltjük a DEBUG-ot, közölve vele, hogy töltse be az XMOD.COM-ot:

DEBUG XMOD.COM

Majd közöljük a DEBUG-gal, hogy konvertálja az XMOD-ot olvasható formába az U-utasítással:

U 100 L XXX

(Az *XXX* az állomány mérete hexadecimálisan).

Mindezek a lépések nagyon nehézkesnek és körülményesnek tűnhetnek, de ha már egyszer megtanultuk, gyorsan és könnyen végrehajthatók, hiszen megismerhetjük, hogy a programozási nyelv hogyan használja az assembly nyelvű interfészrutinokat.

A következő fejezet megismétli e gyakorlat kulcslépéseit, amint bemutatjuk egy rövid, de teljes assembly nyelvű program alkotásának mechanikáját.

Assembly programok írása és összeszerkesztése

Egy assembly program írását és összeszerkesztését egy hihetetlenül egyszerű, de mégis hasznos program írásával mutatjuk be, amely megszólaltatja a számítógép hangszóróját. Ehhez bármely PC-család komputerén (vagy bármely DOS-komputeren) csak a csengő karakterkimenetet (**CHR$(7)**) kell a képernyőre „írni". Ebben a példában a DOS 2. szolgáltatással tesszük ezt,

amelyet a 33-as megszakítás révén hívunk meg. Ezután befejezzük a programot, és visszaadjuk a programvezérlést a 32-es megszakítás használatával a DOS-nak. Ezt a példát követve sok minden megtudható az önálló assembly nyelvű programok írásáról. A program forráskódja:

```
; DOS generic beep program
BEEPSEG    SEGMENT BYTE PUBLIC 'PROG'
           ASSUME   CS:BEEPSEG
BEEP       PROC
           MOV      DL, 7        ;csengőkarakter
           MOV      AH, 2        ;kimeneti karakterfüggvény
           INT      33           ;DOS függvénymegszakítás
           INT      32           ;visszatérés a DOS-megszakítással
BEEP       ENDP
BEEPSEG    ENDS
           END
```

Mint láthatjuk, a program csak négy utasítás hosszú, és csak 8 bájtot foglal el. A következő lépéssel állíthatjuk össze a programot:

MASM BEEP;

Az MASM-utasítás egy tárgyállományt alkot, amely szerkesztésre kész. Ekkor a programot a szubrutinok könyvtára (vagy egyéb tárgykódú állományok nélkül) szerkesztjük:

LINK BEEP;

A szerkesztőprogram rendszerint azt várja, hogy talál az általa szerkesztett programokban egy veremszegmenst, de a mi nagyon egyszerű programunkban nincs. Ez lehetővé teszi, hogy azt egy .COM-állományra konvertáljuk, mint ezt hamarosan látni fogjuk. A szerkesztő reklamálni fog, de figyelmen kívül hagyhatjuk a reklamációját.

A bekapcsolás egy végrehajtható, BEEP.EXE-nek nevezett programot ad. Nagyon általános, hogy az assembly nyelvű programokat úgy írjuk, hogy azokat egy tömörebb .XOM formátumba lehessen konvertálni. Ezt az egyszerű programot a DOS EXE2BIN-utasítás használatával konvertáljuk:

EXE2BIN BEEP.EXE BEEP.COM

Most van egy csengőprogramunk, amely használható bármely számítógépen, amelyiken a DOS fut. Érdemes megállni és megjegyezni, mi történik programunk méretével, amint egy ötletből egy végrehajtható .COM-állománnyá alakul. A program forráskódja kb. 378 bájt (pl. a megjegyzésekben használt helyközöktől függően). Amikor lefordítjuk, felfedezzük, hogy 8 bájtnyi gépi utasítás jön létre. Ez a tárgyállomány, ami overheadként bizonyos szabványos szerkesztési információt foglal magába, 54 bájtos. Sokkal kisebb, mint a forrásállományunk, de nagyobb, mint a tényleges 8 bájt. Szerkesztés után az 54 bájtos tárgykódú állomány 520 bájtos .EXE-állománnyá duzzad. Ez azért van, mert a .EXE-formátumban van egy előtag, amely leírja, hogyan kell a programokat betölteni, és ez az előtag 512 bájtos rekordokat alkot. Ebben az esetben az 512 bájtos overheadet utasításaink 8 bájtja követi. A program .COM-formátumra való konvertálása kiküszöböli az 512 bájt overheadet: egy .COM-állományhoz jutunk, amely csupán 8 bájt tiszta gépi kód.

Az értelmezős és a fordítós BASIC

A BASIC-kel és a BASIC illesztésével való munka rendkívül összetett, itt sem tudunk minden szükségest elmondani; egymaga több könyvet tölthetne meg. A BASIC-kel való szerkesztés különösen összetett feladat, amelyet a kiterjedt PC-család különböző típusainak számos BASIC-változata csak még zavarosabbá tett. A családfa IBM gyártású ágán belül húsznál is több BASIC-változat van. Itt azokkal a kérdésekkel foglalkozunk, amelyek a BASIC-programok külső rutinokhoz való illesztésére vonatkoznak. Két témára összpontosítunk: a BASIC formátumaira (amelyek azért vonatkoznak a külső rutinokra, mert képeseknek kell lenniük a BASIC-kel adatot cserélni) és az interfész konvenciókra (ezek határozzák meg azt, ahogy a BASIC és az assembly nyelv beszél egymással).

A BASIC adatformátumai

A BASIC négy adatformátumot használ: egész számokat, változtatható hosszúságú stringeket, valamint a hosszú és rövid formátumú lebegőpontos számokat, amelyeket a BASIC terminológiában szimpla vagy dupla pontosságú számokként ismerünk. A BASIC-változóknak közvetlenül meg lehet adni a négy formátumtípus valamelyikét, a változónévhez való azonosító toldalék hozzáadásával: % az egészt, ! a szimpla pontosságú (rövid) lebegőpontost, # a dupla pontosságú (hosszú) lebegőpontost és $ a stringet jelenti. A konstansokat hasonlóképpen lehet osztályozni. Az implicit típusadást a **DEF**-utasítással lehet vezérelni, az alap a szimpla pontosság. Hivatkozásként néhány egyszerű példa:

A% Integerváltozó
A! Szimpla pontosságú változó
A Dupla pontosságú változó
A$ Stringváltozó
1% Integerállandó
1! Szimpla pontosságú állandó
1 Dupla pontosságú állandó
"1" Stringállandó

✱ FONTOS MEGJEGYZÉS: *A három numerikus adatformátum mind a két BASIC-ben ugyanaz, a stringformátumok viszont eltérőek.*

Egész számok (integer) adatformátuma

Az egész számok adatformátuma egy szabványos 16 bites, előjeles, 2 bájtos szó, amelyet a PC-családnál általánosan használnak. Az értéket −32 768-tól a 0-án keresztül +32 767-ig terjednek. (☛ Az adatformátum általános tárgyalását l. a 2. fejezetben.) A BASIC nem helyezi el itt a szabványos változatot: az előjel nélküli 16 bites egészt, amelynek értékei 0-tól 65 535-ig terjednek. Mivel az előjel nélküli egészformátum alapvető a PC-ben a memóriacímzéshez és a címszámításhoz, bizonyos elővigyázatossággal kell eljárnunk, amikor a címeket kezeljük. Szokásos a BASIC előjeles egészformátumát használni az előjel nélküli egész számú címek tárolására; ezt részesítjük előnyben, amikor a címeket adjuk át az assembly interfészeknek. Óvatosan kell eljárnunk akkor, amikor olyan előjel nélküli címeket jelzünk ki vagy számítunk,

amelyeket előjeles egészformátumban tárolunk. A rossz számítás megelőzésére az a leghasznosabb, ha a címszámításokat dupla pontossággal, a hosszú lebegőpontos formátum használatával végezzük.

Ha az előjel nélküli szócímek tárolására és átadására a BASIC integer (egész szám) formátumát használjuk, van néhány érdekes pont, ahol vigyázni kell. Elvben a BASIC egész szám formátuma nem tud fogadni semmilyen számot a 32 767 érték határán túl; pl. nem adhatunk egy 50 000-es tízes számrendszerbeli konstansot, vagy egy lebegőpontos változónak egy ezzel egyenértékű értéket. A hexadecimális formátumot használhatjuk 0-tól 65 535-ig értékek adására (vagy &HO-tól &HFFF-ig a BASIC jelölésében). Ez azt jelenti, hogy a BASIC nem engedi meg az I% = 50 000, de megengedi az ezzel egyenértékűt hexadecimálisban: I% = &HC350. A BASIC része a hexadecimális számfüggvény, a **HEX$**, megkönnyítve számukra a decimális értékek hexadecimálisra konvertálását. Ez a két jellemző ésszerű és kényelmes a hexadecimális jelölésben megadott címekkel való munkához.

Biztonságosan konvertálhatunk címértékeket egészből lebegőpontos formátumba. Ezen egyszerű aritmetikai műveletek végrehajtására használjuk a következő módszert:

```
IF I% < 0 THEN D# = I% + 65536# ELSE D# = I%
```

ahol is az I% egy egész, és a D# annak egyenértékűje dupla pontossággal. A dupla pontosságú formátumból egész számba való címérték konvertálásra a következő eljárást használhatnánk:

```
IF D# > 32767 THEN I% = D# - 65536 ELSE I% = D#
```

A **VARPTR** BASIC-függvény közvetlenül nyújtja az integer formátumú változó eltolási címeit (rövidesen látni fogjuk, hogy nyújtja azokat a lebegőpontos változókra is; a stringváltozókra a **VARPTR** közvetett kapcsolatot teremt a változó címéhez). A **VARPTR** a címeltolást a BASIC alapadatszegmensén belül adja, ezt használhatjuk a változó értékének kiolvasására, ill, beírására **PEEK**- vagy **POKE**-utasítással. A következő példával mutathatjuk ezt be:

```
I% = 999        'vagy bármely érték
I.POINTER = VARPTR (I%)
J% = PEEK (I.POINTER) * 256 + PEEK (I. POINTER + 1)
PRINT I%, J%
```

Egy változó címeltolási értékének kiszámítására ennek a példának kevés gyakorlati haszna van, de bemutatja, hogyan találhatjuk meg és használhatjuk a címeket. Az ehhez hasonló példákat azért terveztük, hogy bizalmat keltsünk, és segítsünk a memóriacímek BASIC-ben való használatának megértését. Az assembly nyelvű interfész készítésekor derül csak ki, hogy mindkettő milyen fontos.

Lebegőpontos adatformátumok

A lebegőpontos értékeket – a szimpla, ill. a dupla pontosságút – a BASIC-re jellemző közös formátumban tároljuk. A BASIC lebegőpontos formátuma más, mint a legtöbb egyéb, PC-ken használt programnyelvé, és nem kompatibilis a 8087 és a 8287 matematikai koprocesszorok által használt formátumokkal.

A BASIC lebegőpontos formátumának hasznosításához leírjuk annak kulcselemeit, de előre figyelmeztetünk: a lebegőpontos formátumok tárgya igen összetett. A következő rész alaposabb ismereteket feltételez a számítógépekben a lebegőpontos számok tárolásáról és használatáról. A BASIC-ben a szimpla és dupla pontosságú adatformátumok csak a mantissza jegyeinek számában térnek el. A formátum egyéb részei, beleértve a mantissza értéktartományát, ugyanaz. A szimpla pontosságú adat 4 bájtot foglal el, a dupla pontosságú adat 8 bájtot.

A mantisszát az első három (vagy 7) bájtban tároljuk, a legkisebb értékű bájtot az első helyen (követve a PC-mikroprocesszorok gyakorlatát). A kitevőt tároljuk utoljára, elfoglalva az utolsó bájtot. A 4, ill. 8 bájtot a következőképpen vázolhatnánk:

M7 M6 M5 M4 M3 M2 M1 E

A kitevőt 2 hatványaként, 128-cal eltolva tároljuk. Ez azt jelenti, hogy a 0 kitevőt 128-ként (hexa 80), a −3 kitevőt pedig 125-ként (hexa 7D) tároljuk.

A mantisszát egy rendezett bináris törtként tároljuk, az első (vagy magas) értékű bitet beleértve. A magas értékű bájt magas értékű bitje használatos az előjel tárolására: 0 jelöli a pozitív és 1 a negatív értékeket. Az előjelbit azt a helyiértéket foglalja el, amely a legnagyobb magas értékű mantisszabit (de azt nem használja).

A következő program a lebegőpontos számok dekódolását mutatja. Jól illusztrálja az előzőekben tárgyaltakat, és ha baj van, segít a formátum megértésében.

```
100 INPUT "Vigyél be egy értéket", SINGLE
110 ADDRESS = VARPTR (SINGLE)
120 PRINT "A hexadecimális bájtok"
130 FOR I = 0 TO 3
140     H = PEEK (ADDRESS + I)
150     IF ≦ H 16 THEN PRINT "0";
160     PRINT HEX$ (H); " ";
170 NEXT
180 PRINT
190 E# = PEEK (ADDRESS + 3)
200 M1# = PEEK (ADDRESS + 2)
210 M2# = PEEK (ADDRESS + 1)
220 M3# = PEEK (ADDRESS + 0)
230 EXPONENT# = E# - 128
240 SIGN = M1# / 128
250 M1# = 128 + M1# MOD 128
260 MANTISSA# = M1# / 256 + M2# / (256 * 256) + M3# / (256 * 256 * 256)
270 VALUE# = MANRISSA# * 2^EXPONENT#
280 IF SIGN THEN VALUE# = -VALUE#
290 PRINT "A dekódolt érték:"; VALUE#
300 PRINT
310 GOTO 100
```

Figyeljük meg, hogy a 190–220-as sorok szimpla pontosságú, 4 bájtos formátumot különítenek el, a mantisszabájt bemutatására használt jelölésnek megfelelő változónevek felhasználásával. Korábban már megállapítottuk, hogy a kitevőt eltolt formában tároljuk; a 230-as sor visszaállítja ezt az eltolást, és −128-tól +127-ig terjedő tartományú kitevőt ad nekünk. A 240-es és 250-es sor gondoskodik a legnagyobb helyértékű mantisszabájt magas értékű bitjeiről: a 240-es előjelként rögzíti a bitbeállítást, a 250-es a beleértett egy bitet a helyére helyezi. Ezután a 260-as rakja össze a három mantisszabájtot egy értékké, amelynek tizedespontja (vagy kettes pontja) pontosan az első bithely előtt van. A 270-es aztán beszorozza a kitevőértéket, a 280-as alkalmazza az előjelet, és ezzel dekódoltuk a BASIC lebegőpontos formátumot.

Miután végeztünk a lebegőpontos formátumokkal, megnézhetjük, hogyan tárolja a BASIC a stringeket.

Stringformátumok az értelmezős BASIC-ben

A stringértékeket két részben tároljuk: az egyikben a stringleírót, amely a string hosszát és eltolási helyét tárolja, a másikban magát a string értéket, amely ASCIIZ-karakterek sorozata.

A stringleíró 3 bájt hosszú. Az első bájt a string hosszát tartalmazza, ez a maximális méretet 255 bájtra korlátozza. A következő két bájt adja az adatszegmens-eltolást a tényleges stringértékhez. A tényleges stringértéknek nincs különleges formátuma; egyszerűen bájtok sorozata a megadott címen.

Amikor a **VARPTR**-függvényt alkalmazzuk egy stringre, akkor az a stringleíró eltolási helyét adja vissza. A stringleíróból megkaphatjuk magának a stringnek az eltolási címét. A következő program mutatja be ezen információ megtalálásának és dekódolásának folyamatát:

```
100 INPUT "Vigyen be egy stringet", OUR.STRING$
110 DESCRIPTOR.ADDRESS = VARPTR (OUR. STRING$)
120 PRINT "A stringmutató hexadecimális címe";
130 PRINT HEX$ (DESCRIPTOR.ADDRESS)
140 STRING.LENGTH = PEEK (DESCRIPTOR.ADDRESS)
150 PRINT "A string hossza:";
160 PRINT STRING.LENGTH
170 STRING.ADDRES = PEEK (DESCRIPTOR.ADDRESS + 1)
    + 256 * PEEK (DESCRIPTOR.ADDRESS + 2)
180 PRINT "A string hexadecimális kezdőcíme";
190 PRINT HEX$ (STRING.ADDRESS)
200 PRINT "A string:";
210 FOR I = 0 TO STRING.LENGTH - 1
220     PRINT CHR$ (PEEK (I + STRING.ADDRESS));
230 NEXT I
240 PRINT : PRINT
250 GOTO 100
```

Stringformátumok a fordítós BASIC-ben

A stringadatok formátuma merőben más a fordítós BASIC-ben. Nincs különbség a megengedett stringek típusában, csupán méretében. Az értelmezős BASIC-ben a string hosszúsága egy előjel nélküli, 1 bájtos egész, amely 0–255 karakter hosszúságot enged meg. A fordítós BASIC a hosszúságot 2 bájtos előjeles egészként ábrázolja, amely 0–32 767 karakter hosszúságot enged meg (a lehetséges negatív értékeket figyelmen kívül hagytuk).

Mindkét típusú BASIC-nél a stringekhez közvetve a stringleírón keresztül férünk hozzá. A változónévhez kapcsolódó memóriacím – a cím, melyet a **VARPTR (NÉV$)** adott – a string helyett a stringleíróra mutat. A stringleíró két mezőből áll: az egyik a string hosszát, a másik a tényleges stringérték adatszegmensbeli eltolását tartalmazza. Az egyetlen különbség az, hogy a mezőhossz a fordítós BASIC-nél 2 bájt az értelmezős BASIC-nél viszont csak 1.

A lefordított stringeket a következő programmal vizsgálhatjuk meg. (Felhívjuk a figyelmet, hogy a program számos, csak a fordítós BASIC-ben megengedett jellemzővel bír, tehát nem vizsgálhatjuk az értelmezős BASIC-kel.) Ez a program az előző részben megadott BASIC-programtól csak a 140-es sorban tér el, ahol felveszi a string hosszát, valamint a 170-es sorban, a stringelhelyezés kiolvasására használt **PEEK**-eltolásokban.

```
100 INPUT "Vigyen be egy stringet", OUR. STRING$
110 DESCRIPTOR.ADDRESS = VARPTR (OUR.STRING$)
120 PRINT "A stringmutató hexadecimális címe";
130 PRINT HEX$ (DESCRIPTOR.ADDRESS)
140 STRING.LENGTH = PEEK (DESCRIPTOR.ADDRESS)
    + 256 * PEEK (DESCRIPTOR.ADDRESS + 1)
150 PRINT "A string hossza:";
160 PRINT STRING:LENGTH
170 STRING.ADDRESS = PEEK (DESCRIPTOR.ADDRESS + 2)
    + 256 * PEEK (DESCRIPTOR.ADDRESS +3)
180 PRINT "A stringérték hexadecimális kezdőcíme";
190 PRINT HEX$ (STRRING.ADDRESS)
200 PRINT "A string értéke:";
210 FOR I = TO STRING.LENGTH - 1
220     PRINT CHR$ (PEEK (I + STRING.ADDRESS));
230 NEXT I
240 PRINT : PRINT
250 GOTO 100
```

Az értelmezős BASIC assembly interfésze

Ebben a részben azokat az interfészszabályokat tárgyaljuk, amelyek az értelmezős BASIC-programokból hívott assembly nyelvű interfészekre vonatkoznak. Csak a **CALL**-lal hívott szubrutinokat tárgyaljuk, az **USR**-függvényeket nem. Az **USR**-függvények bosszantó és szükségtelen komplikációkat vonhatnak maguk után, és olyan problémákat rejtenek magukban, amelyek nagyon eltérőek az IBM és a nem-IBM gépeknél. Általában nem javasoljuk az **USR**-függvények használatát.

Az értelmezős BASIC egy szabványos interfészkonvenciót használ, amely az assembly szintű nyelvi kapcsolatok szokásos mintájába jól beleillik.

A konvenciók lényege:

- Minden paramétert úgy adunk át, hogy az eltolási címeiket a verembe helyezzük, ezt a formátumot „név szerinti hívás” formátumként ismerjük. Ez azt jelenti, hogy az assembly nyelvű szubrutinok mind hozzáférhetnek és megváltoztathatják a paraméterek értékeit; ill. azt is jelenti, hogy a veremben címek, és nem értékek jelennek meg, tehát az értékekhez való hozzáférés több lépést igényel.
- A paramétereket az írás sorrendjében adja át – úgy, mint a legtöbb más nyelvnél is. (☛ Kivéve a C nyelvet, l. a következőkben). Ez azt jelenti, hogy az első paraméter közelebb van a verem aljához, az utolsó paraméter pedig a tetejéhez.
- A szubrutinokat a **FAR CALL**-lal hívjuk meg, ezért a **FAR RET**-utasítással kell befejeződniük.
- A szubrutin felelős azért, hogy a paramétereket a veremből eltávolítsa – ugyanúgy, mint a legtöbb más nyelvnél. (☛ Kivéve a C nyelvet, l. később.) A szubrutinoknak egy előre ismert fix paraméterszámot kell adni.
- Az *AX*-regiszter, más nyelvektől eltérően, nem használható hibakódok vagy értékek visszaadására. A szubrutin bármely információt a paramétereken keresztül ad vissza.
- A BASIC állítja be és tartja karban a szegmens-, kód- és veremregisztereket. A többi regisztert szükség szerint lehet változtatni.

Egy assembly nyelvű interfészrutin részleteit a korábban e fejezetben (és a 8.-ban) már tárgyalt öt logikai szint szerint fogjuk leírni. Az 1. és 2. szintet (az assembler overheadet) a következőképpen kódolhatjuk:

```
MY_SEG   SEGMENT
         ASSUME
MY_PROG  PROG       CS:MY_SEG
                    FAR
               ; itt jelenik meg a 3–5. szint
MY_PROC  ENDP
MY_SEG   ENDS
END
```

A MY_SEG és MY_PROC nevek tetszőlegesek. Az értelmezős BASIC-ben bármely elfogadott nevet választhatunk a szubrutin azonosítására, a fordítós BASIC-ben viszont vannak megkötések.

Ha ezt a rutint összehasonlítjuk a 8. fejezetben tárgyalt általános interfészrutinnal, két különbséget vehetünk észre. Ebben a példában a **SEGMENT**-utasítást nem követi "**CODE**" osztály, és nincs **PUBLIC MY_PROC** állapot. Mind a két változás, és az a tény, hogy az eljárási név lényegtelen a BASIC-ben, azzal függ össze, hogy az assembly nyelvű interfészeket nem szerkesztik össze az értelmezős BASIC-kel. Azoknál a programoknál amelyeket öszeszerkesztünk, akárcsak az e fejezetben tárgyalt összes többi nyelvnél, ezek lényegesek.

A következő figyelembe veendő szint (3), adja a szubrutinnak a belépési és kilépési kódokat. A kód formája a következő:

```
PUSH   BP
MOV    BP,SP
;itt jelenik meg a 4. és 5. szint
POP    BP
RET    XXX
```

Mind a négy utasítás szabvány, és ugyanannak kell lennie az öszes assembly nyelvű interfésznél. Egyetlen változó, amelyet ebben a példában *XXX*-szel jelöltünk, képviseli a veremből eldobandó paraméterbájtok számát. Az *XXX* értéke kétszerese a BASIC **CALL**-szubrutin behívásakor átadott paraméterek számának, mivel minden egyes paraméter a veremben 2 bájt. Ha pl. három paramétert adtunk át, akkor a verem kitakarításhoz az *XXX*-értéke 6; ha nincs paraméter, akkor pedig 0. Egyébként az ebben a példában használt **RET**-utasítás egy **PROC FAR**-utasítás segítségével (amely a rutin elején jelent meg) **FAR RET**-utasítássá fordítódik le.

A következő (a 4.) szint vonatkozik a paraméterek elérésére és megváltoztatására. Mint ahogy azt már megbeszéltük, az utolsó paraméter címe a veremben a [*BP* + 6]-ként hivatkozott helyen lesz, az előtte levő paraméter a *BP* + 8 (és így tovább).

Példaként tegyük föl, hogy három paraméterünk van, mind a három egész szám, és az *AX*-, a *BX*- és a *CX*-regiszterekbe akarjuk tölteni azokat (bármilyen oknál fogva). Itt következik egy 4. szintű kód, amely ezeket a paramétereket veszi elő:

```
MOV   SI,[BP+10]  ;az első paraméter címének lekérdezése
MOV   AX,[SI]     ;az első érték lekérdezése
MOV   SI,[BP+8]   ;a második paraméter címének lekérdezése
MOV   BX,[SI]     ;a második érték lekérdezése
MOV   SI,[BP+6]   ;a harmadik paraméter címének lekérdezése
MOV   CX,[SI]     ;a harmadik érték lekérdezése
```

Figyeljük meg, hogy a paraméterértékeket két lépésben kapjuk: először megkapjuk a címet (amit az *SI*-regiszterbe teszünk, mivel ezt kényelmes használni), és azután a címet használjuk arra, hogy megkapjuk az értéket, amelyet oda helyezünk, ahová ténylegesen akarjuk.

Ez csak a 4. szint első felét mutatja be. A második fele adja vissza a BASIC-programnak az értékeket úgy, hogy azokat a veremben tárolja. Ekkor lekérdezzük a paraméter címét, majd erre a címre eltároljuk az új értéket. Tegyük fel pl., hogy kiszámítottunk egy új értéket, és az a *DX*-regiszterben van. Vissza akarjuk adni a BASIC-nek mint az első paraméter új értékét. Ezt két egyszerű utasítás használatával tehetjük:

```
MOV    SI,[BP+10]  ; az első paraméter lekérdezése (újból)
MOV    [SI],DX     ; az érték visszaadása a DX-ből
```

A teljesség kedvéért következik egy assembly nyelvű szubrutin, amely az *SS*- és az *SP*-regiszterek értékeit adja vissza. Ez lehetővé teszi a BASIC-program számára, hogy saját vermét megvizsgálja. Általában nem lenne ok ennek elvégzésére, de nagyon tanulságos lehet. A szubrutin feltételezi, hogy két integer típusú paraméterrel hívják; az aktuális veremszegmens (*SS*) és a veremmutató (*SP*) értékeket ezekbe a paraméterekbe helyezi.
Következik a rutinunk:

```
STACKINFO    SEGMENT
             ASSUME    CS:STACKINFO
GETSTACK     PROC      FAR
             PUSH      BP
             MOV       BP,SP
             MOV       SI,[BP+8]
             MOV       [SI]?ŚŞ
             MOV       SI,[BP+6]
             MOV       [SI],SP
             POP       BP
             RET       4
GETSTACK     ENDP
STACKINFO    ENFDS
END          END
```

A fordítós BASIC assembly interfésze

Bárki, aki a PC-család gépein kiterjedten dolgozott BASIC-kel, kétségkívül ismer néhány bosszantó különbséget és inkompatibilitást, amely fennáll a két BASIC között. Itt mi csak azt a jelentős különbséget jegyezzük meg, amely az assembly nyelvű interfész készítésére vonatkozik. Néhány példát is mutatunk.

Az értelmezős BASIC-ben az assembly nyelvű interfésszel való munkában az a szokásos, hogy a rutint valamilyen memóriahelyre helyezzük (ez a legbosszantóbb hibákat előidéző programozási munka a BASIC-ben). Az interfész címének memóriaeltolására beállítunk egy változót, és a CALL-utasítással hívjuk a változónevet.

Ezzel ellentétben a fordítós BASIC két elsődleges eljárást ad az assembly nyelvű interfészekkel való munkára. Az egyik szorosan illeszkedik az értelmezős BASIC-eljáráshoz, a másik követi a szabványos fordítós nyelvekre vonatkozó opciókat.

Az első eljárás szorosan követi az értelmezős BASIC-eljárás lényegét, bár attól formájában különbözik. Bemutatásukra legyen a LOCATION% a BASIC-változó neve, amelyet az assembly nyelvű interfész memóriaeltolási helyére teszünk, és legyen PARAMETERS bármely, az interfésznek átadott paraméter. Ezzel a jelöléssel az interfészt a következőképpen használnánk:

CALL LOCATION% (PARAMETERS)

A fordítós BASIC-ben az interfész elérési módja eltérő formátumú, bár a végeredmény pontosan ugyanaz:

CALL ABSOLUTE (PARAMETERS, LOCATION%)

A fordítós BASIC-változatban kritikus különbség, hogy a LOCATION%-változónak egésznek kell lennie. Az interpreteres BASIC-ben bármilyen számváltozó elfogadható.

Érdemes ennek a **CALL ABSOLUTE**-műveletnek néhány mechanizmusát megnézni. A fordítós BASIC-ben az összes hívás ténylegesen konvencionális hívás külső rutinokhoz, amelyek kapcsolódnak a lefordított programokhoz. Ez a szabványos mechanizmus, amelyet az összes fordítós nyelv használ. A fordítós BASIC lényegében az értelmezős BASIC **CALL LOCATION%**-ával megegyező dolgot szimulál, egy ABSOLUTE-nak nevezett szubrutin-könyvtár segítségével. Más szóval, az ABSOLUTE nem olyan része a BASIC nyelvnek, mint az **ON ERROR** vagy a **CHAIN**. Ehelyett az ABSOLUTE egyszerűen egy BASIC-könyvtárban levő külső szubrutin neve. Az ABSOLUTE-szubrutin egy egyszerű műveletet hajt végre: a paramétereit arra használja (beleértve a LOCATION% paramétert), hogy szimulálja a **CALL LOCATION%**-műveletet, amelyet az értelmezős BASIC végrehajt.

Mielőtt továbbmennénk a fordítós BASIC másik, assembly nyelvű interfészeket kezelő módszerére, fel kell hívni a figyelmet azokra a problémákra, amelyek az assembly nyelvű interfész memóriabeli elhelyezésekor adódnak. A mechanizmus, amelyet arra használunk, hogy ezeket az interfészeket memóriába helyezzük, ugyanaz mind a két BASIC-ben: általában vagy **BLOAD**-dal töltjük be az interfészt egy állományból, vagy a **POKE**-utasítással bájtonként visszük azt a memóriába. Bár a mechanizmusok azonosak mind a két BASIC-nél, a megfelelő memóriahely megtalálásának módszere más és más lehet, a körülményektől és a programozási technikától függően. Észre fogjuk venni, hogy eddig sikeresen kikerültük e témát. Ezúttal is így teszünk, mert rendkívül zavaros –, valóban a BASIC-könyvek különleges területe.

A fordítós BASIC-ben az assembly nyelvű interfészek kezelésére használt másik módszer, a programozási nyelvekben általánosan elfogadott. A modulokat (mint assembly nyelvű interfészeket) külön készítjük el, és vagy egy külön tárgykódú állományban (amelynek névkiterjesztése .OBJ), vagy egy tárgykódú könyvtárban (melynek állománynév kiterjesztése .LIB) tároljuk. Mindkét esetben a különálló modulokat (azaz a fordítós BASIC-programokat és az összeszerkesztett assembly nyelvű interfészeket) az előző fejezetben leírtak szerint szerkesztjük össze a LINK-programmal.

Nézzük meg mindkét oldalról, hogyan használjuk ezt az interfészrutint! A BASIC oldaláról egy assembly nyelvű interfészt a következőképpen hívunk meg:

CALL NAME (PARAMETERS)

A paraméterek ugyanazok, mint az előző módszerben vagy az értelmezős BASIC-ben. A név ugyanaz a név, amit az assembly nyelvű interfészben láttunk, és ami a kívánt rutint azonosítja. A lépések (amelyeket a fordítós BASIC a szubrutin hívásában követ) ugyanazok, mint amelyeket az értelmezős BASIC követ; ugyanazok az interfészszabályok vonatkoznak a verem használatára, a paraméterek verembe helyezésére és a hívásra (**FAR CALL**, amely **FAR RET**-utasítást kíván). Emiatt az előző értelmezős BASIC-re vonatkozó rutint használjuk (azt, amely az aktuális veremszegmens- és a veremmutató-beállításokat adja át.) A különbségekre felhívjuk a figyelmet:

```
STACKINFO SEGMENT
          ASSUME        CS:STACKINFO
          PUBLIC        GETSTACK
```

```
GETSTACK  PROC   FAR
          PUSH   BP
          MOV    BP,SP
          MOV    SI,[BP+8]
          MOV    [SI],SS
          MOV    SI,[BP+6]
          MOV    [SI],SP
          POP    BP
          RET    4
GETSTACK  ENDP
STACKINFO ENDS
          END
```

Egy ilyen assembly interfésznek fordítós BASIC-programhoz való szerkesztésekor fellépő új igény, hogy a szubrutin nevét nyilvánossá kell tenni: azaz deklarálni kell azt egy assembler által használandó névként. Ezt a rutin harmadik sorában tesszük, amely a következőképpen néz ki: **PUBLIC GETSTACK**. Ez az egyetlen különbség e rutin és az előző között. (Valójában a **PUBLIC** nem okozna semmilyen kárt az előző rutinban, de nem volna célja, mert az értelmezős BASIC nem használja azt.)

Az ilyenfajta assembly nyelvű interfész bemutatására vegyünk egy fordítható BASIC-programot, amely ezt használja, és az eredményt kijelzi:

```
100 ' a szerkesztett interfész használatának bemutatása
110 '
120 I% = 0
130 J% = 0
140 CALL GETSTACK (I%, J%)
150 PRINT "Egy program közepén"
160 PRINT "a veremregiszter: "; HEX$(I%)
170 PRINT "a veremmutató:" ;HEX$(J%)
180 GOSUB 200
190 GOTO 270
200 ' a go-sub alrutin
210 '
220 CALL GETSTACK (I%,J%)
230 PRINT "A GO–SUB alrutinban"
240 PRINT "a veremszegmens:" ;HEX$(I%)
250 PRINT "a veremmutató:" ;HEX$(J%)
260 RETURN
270 END
```

Számos említésre méltó van ebben a programban. Amikor az assembly nyelvű alprogramot hívjuk (a 140-es és 220-as sorok), a nevével hívjuk. **GETSTACK** az a név, amely az assembly nyelvű kód **PUBLIC**-utasításában megjelenik. A neveket a szerkesztőprogramnak fel kell ismernie. Az assembly nyelvű programban levő szegmensnév (STACKINFO) önkényes, ezért hossza nem számít, mivel csak itt használjuk. Tehát jegyezzük meg, hogy mind a hívó BASIC-programnak, mind a hívott assembly nyelvű programnak azonos számú paramétert kell használni. A **RET 4** nevű utasítás, amely az utolsó működő utasítás, a paraméterek 4 bájtját veszi a veremből. Ez két paraméternek felel meg, mivel egy paraméter 2 bájtos. (Egy kifinomult és trükkös assembly nyelvű rutin elfogadhat változó paraméterszámot, de az eljárás számunkra túl bonyolult.)

Ez a mintaprogram kijelzi kétszer a verem állapotát, ahogy azt az assembly nyelvű rutinban észleli: egyszer az utasítások egyenes vonalán és egyszer a **GOSUB**-rutinban. A két

veremmutató értékének különbsége mutatja, hogy a GOSUB-ok használata hogyan hat a BASIC-veremre. Ezt a példát futtatva megismerhetjük a választ.

Kövessük nyomon, milyen lépéseket kell tennünk az előző példában szereplő fordítós BASIC-program és az assembly nyelvű interfész elkészítéséhez! Tegyük fel, hogy az assembly nyelvű forráskódunk egy GETSTACK.ASM-nek nevezett állományban van! A következő utasítással kell lefordítanunk:

```
MAMS GETSTACK:
```

Ez egy GATSTACK.OBJ-nek nevezett tárgykódú állományt hoz létre. Tegyük fel, hogy a BASIC forráskódunk egy TEST.BAS nevű állományban van, amelynek ASCII-állománynak – és nem BASIC-állománynak – kell lennie. A következő utasítással fordítjuk le:

```
BASCOM TEST;
```

Ez egy TEST.OBJ nevű tárgykódú állományt hoz létre. A kettőt a következő utasítással szerkesztjük össze:

```
LINK TEST+GETSTACK
```

Ez egy TEST.EXE nevű programállományt ad kimenetként.

A Pascal nyelv

Ebben a részben az IBM PC Pascal fordítóját és a család másik tagját, a Microsoft Pascal-t fogjuk tárgyalni. Amikor a Pascal-adatformátumokat tárgyaljuk, általában jelezni fogjuk, melyik e fordítók sajátjai, és melyek a szabványos Pascal tulajdonságai. Ha e fordítók bármely változatát használjuk, az itt megadott információk a részletekbe menően is alkalmazhatók; az itteni leírást alapul véve saját fordítónk jellemzőit is megállapíthatjuk.

Néhány jelentős eltérés van az IBM PC Pascal fordítójának 1. és 2. változata között. Ezekre akkor térünk ki, amikor az adatformátumokat tárgyaljuk.

A Pascal adatformátuma

A Pascal nyelv három ismerős adatformátumot használ: integer, lebegőpontos (a Pascal terminológiában REAL-ként ismert) és string. Van egy különleges típus is, amelyet setként ismerünk. A típusokon belül jó néhány változat van, különösen az integertípuson belül.

Az egész számok

Kezdetben az összes egész (integer) adattípussal foglalkozunk. Az integerek bináris számokként helyezkednek el a memóriában. Az integer lehet 1, 2 vagy 4 bájt hosszú, előjeles vagy előjel nélküli szám. Az alacsony helyiértékű bájt található a kisebb címen. E leírás által javasolt hat formátumból ötöt használunk ténylegesen, a 4 bájtos előjel nélküli nem használt, némelyik formátum pedig kettős feladatot lát el.

Az egybájtos integerek előjeles és előjel nélküli formában állnak rendelkezésre. Az előjeles forma tartománya −128-tól +127-ig terjed, és a Pascalban SINT-nek hívjuk (nyilvánvalóan a short integerből); a nem előjeles tartománya pedig 0-tól 255-ig, és BYTE-nak hívjuk. Sem a SINT, sem a BYTE nem része a szabvány Pascalnak. Mivel nem túl sokat tárgyalják a fordító dokumentációjában, könnyű átsiklani felettük.

A kétbájtos integerek ugyancsak rendelkezésre állnak az előjeles és az előjel nélküli formában is. Az előjeles formát INTEGER-nek nevezzük, értelmezési tartománya −32 768-tól +32 767-ig terjed, ez egy szabvány Pascal-formátum. Az előjel nélküli alak értelmezési tartománya 0-tól 65 535-ig terjed, WORD-nek hívjuk: ez nem szabvány Pascal-formátum. Az INTEGER-formátum természetesen a legáltalánosabb formátum a PC programozási nyelvei között; pontosan megegyezik pl. a BASIC által használt formátummal. Az előjel nélküli 16 bites WORD-formátumnak saját jelentése is van. A Pascalban levő összes címzési lehetőség – az **ADR** és az **ADS** – a WORD-formátumon alapul. (☛ A Pascalban levő címzésekre vonatkozóan a további megjegyzéseket l. az interfész tárgyalásánál). A Pascal WORD-formátuma ugyanaz, mint a PC-családban használt szabványos, előjel nélküli 16 bites integerek, és ugyanaz, mint a C-ben használt UNSIGNED INT-formátum.

A négybájtos integerek csak előjeles formában állnak rendelkezésre, és csak a 2.0 változatszámú vagy a későbbi fordítókban. E formátum tartománya a −2 147 483 647-tól a +2 147 483 647-ig terjed, és INTEGER4-nek hívjuk. (Vegyük észre, hogy kellene (valószínűleg van is) egy további negatív érték, bár a fordító által meghatározott formátumként nincs. Ha ilyen kell, akkor valószínűleg mindenképpen baj van.) Ez a 32 bites integer nem része a szabványos Pascalnak, és sok más PC-nyelv sem használja. Mindamellett része a C-nek, ahol ezt LONG INT-nek (l. később.) hívják. Az INTEGER4-formátum nincs beillesztve a többi Pascal integer-formátumai közé, sok helyen, ahol más integerformátumok használhatók, az INTEGER4 nem használható.

A szabványos Pascal nyelv az integerek általánosítását nyújtja a felsorolás (enumeration) nevű adattípusban. Gyakorlati célokból minden ilyen felsorolásnak saját neve van, és előjel nélkül integereket tartalmaz 0-tól felfelé. A tényleges adatformátum az öszes felsorolás-adattípusra vonatkozóan az előjel nélküli integer 1 bájtban (BYTE) vagy 2 bájtban (WORD), attól függően, hogy a számérték befér-e egy bájtba. A BOOLEAN a leggyakrabban használt speciális típus, egyszerűen egy előre deklarált, kétértékű felsorolástípus. (A BOOLEAN a nyelvi szinten definiált logikai változó. Értelmezett értékei: true (igaz) és a false (hamis). A BOOLEAN típusú változókkal az értékadás és a logikai műveletek vannak értelmezve. A lektor.) Mint ilyen, egy bájtot foglal el, és a 0 vagy az 1 értéket veszi fel.

String formátumok

A szabványos Pascalnak van egy karaktertípusa, (CHAR-nak hívják) és egy egyetlen bájtban tárolt egyetlen ASCII-karaktert tartalmaz. Szempontunktól függően a CHAR-t önálló adattípusnak, a bájttípus speciális esetének, vagy a string adattípusok egy elemének tekinthetjük. Mindamellett a Pascal a CHAR-t mint saját megkülönböztetett formátumát kezeli.

Két stringformátum van: egyik része a szabványos Pascalnak, melyet stringnek nevezünk, és fix hosszúságú karakterstringeket tartalmaz; a másik a Pascal bővítése, melyet LSTRING-nek nevezünk, és változó hosszúságú karakterstringeket tartalmaz. A Pascal-fordítók többsége a szabványos nyelvet kibővítette a változtatható hosszúságú stringekkel. Ezt azonban különböző módon tették. Mi az IBM/Microsoft fordítók által használt formátumot tárgyaljuk.

A fix hosszúságú stringeket egyszerűen a bájtok stringjeként tároljuk, amelyeknek nincs külön lezáró karakterük vagy formátumuk. Jegyezzük meg, ha az S egy fix hosszúságú string, akkor a Pascal jelölésben a string első karaktere S[1]. A fix hosszúságú string címe ugyanaz, mint az első karakterének címe. A változtatható hosszúságú stringeket karakterekként tároljuk, amelyet egy előjel nélküli integer formátumú (e fordító jelölésében 1 bájt hosszúságú) kód előz meg; ezért a változtatható hosszúságú string 0-tól 255 karakterig terjedhet. A változtatható hosszúságú stringnek (ugyanúgy mint a fix hosszúságúnak) S[1] az első karaktere. Az S [0]

hivatkozik arra a bájtra, amelyik a string aktuális hosszát tartalmazza. A változtatható hosszúságú string címe a hosszúság bájtjának (S[0]) címe – az első adatbájt (S[1]) címe helyett.

MEGJEGYZÉS: Bármely string hosszát a Pascalban deklarálni kell. A fix hosszúságú stringnél a hosszúság megegyezik a stringben levő bájtok számával; a változtatható hosszúságú stringek egy kiegészítő bájtot használnak.

A halmaz adatformátum

A halmaz (SET) típus Pascal különlegessége, kevés más nyelvben található meg. A halmazok egy felsorolástípuson alapulnak a felsorolásnak legfeljebb 256 eleme lehet. A halmazokra vonatkozó adatformátum minden egyes felsorolásbeli elem számára egy külön bitet jelöl ki. Ha a felsorolásbeli elem a halmaz része, akkor a megfelelő bit 1-re állítódik. A halmazadatok mérete attól függ, hogy hány bitre van szükség; azaz attól, hány elem van a felsorolásban. Minden várakozásunk ellenére az adattípus mérete mindig legalább 2 bájt, így egy 8 elemű halmaz is 2 bájtot foglal el – bár 1 bájt is elég lenne. A halmaz minimális mérete 2 bájt (ha az elemek száma 1 és 16 közé esik); maximális mérete 32 bájt (ha az alapként szolgáló felsorolásban az elemek száma 241-től 256-ig terjed).

A halmazban levő biteket balról jobbra haladva jelöljük ki, és a bitkódolás megfelel a felsorolásban deklarált elemek felsorolásbeli sorrendjének. A katalógustípusban levő első elem pl. megfelel az első bájt első magasértékű bitjének (hexa 80); a következő elem a katalógustípusban a következő bitnek (hexa 40) és így tovább. A nem használt bitek – amelyek az adat méretét 2 bájtra vagy 16 bitre kerekítik – 0-ra állítódnak, mint ahogy az várható.

Lebegőpontos adatformátumok

A lebegőpontos adatformátumoknak, amelyeket a Pascal terminológiában REAL-formátumnak hívunk, van néhány érdekessége. Három formátumot kell figyelembe vennünk. Az első formátum a fordító első változatában van, és pontosan megegyezik a BASIC által használt szimpla pontosságú számokkal. Tárgyalásunkban ezt a formátumot egyszerűen csak régi real-formátumnak fogjuk nevezni. A másik két formátum a fordító 2. (és későbbi) változataira vonatkozik; ezek a formátumok megegyeznek azzal a formával, amelyet a 8087 matematikai koprocesszor is használ. Ezt a két formátumot a Pascal-fordító REAL4-nek és REAL8-nak nevezi, a továbbiakban mi ezeket új real-formátumnak hívjuk.

MEGJEGYZÉS: Nincs egységes módszer a régi real-formátum adatoknak új real-formátumra alakítására.

Először a régi real-formátumot beszéljük meg röviden, hogy összehasonlíthassuk azt a másik két formátummal. (☛ A további részleteket l. a BASIC szimpla pontosságú formátumának tárgyalásánál, mivel a két formátum ugyanaz.) Feltételezzük a lebegőpontos aritmetika és a tárolásukra alkalmazott adatformátumok ismeretét; e lebegőpontos adatformátumok sajátosságait fogjuk tárgyalni, nem pedig a lebegőpontos számok alapelveit.

A régi real négybájtos formátum, amelyet a következőképpen jellemezhetünk:

M3 M2 M1 E

Az M bájtok tárolják a mantisszát (az alacsony értékű bájtokat tároljuk először, az M1 magasértékű bitje tartalmazza az előjel értéket az 1 mutatja a negatív jelet). Az E bájt tartalmaz-

za a bináris kitevőt 128-as eltolással (azaz E-128 adja az igazi kitevőt). A mantisszát törtként kezeljük, azaz a „tizedes" pont (valójában kettes pont) a benne foglalt nagyértékű bittől balra esik.

Az új real-formátumok minden lényeges részletben különböznek a régi real-formátumtól. Egymás között ugyanazt a tervezést követik, csak az egyes részeknek fenntartott bitek számában térnek el egymástól.

Minden egyes formátum három mezőből áll: előjel, kitevő és mantissza. Ez a tárolási sorrend, elöl a legalacsonyabb értékű bitek vannak. Más szóval az utolsó bájt legmagasabb helyiértékű bitje vagy a jobboldali első bájt tartalmazza az előjelet, a következő magasértékű bit a kitevőt (és így tovább). Ezt a mintát követve a mantissza legalacsonyabb értékű bitjét az első bájtban találjuk. Az előjelmező mindkét formátumban 1 bit, amelyben 1 jelöli a negatív és 0 a pozitív számokat.

A kitevőmező eltolásos, integer formátumú. A REAL4 esetében a kitevő 8 bitet tölt meg 127-es eltolással (azaz 127-et levonva) adja az igazi kitevő értéket. A REAL8-ban a kitevő 11 bitet foglal el, 1 023 eltolással. Ezért a REAL4 kitevői −127-től +128-ig, a REAL8 kitevői pedig −1 023-tól +1 024-ig terjednek. Vegyük észre az összehasonlításnál, hogy a régi real-formátumok (amelyeket a BASIC is használ) 128-as eltolásúak – az itt használt 127-essel szemben! Az új real-formátumok kissé jobbak az igen nagy számokra, míg a régi real-formátumok az igen kicsi számokra. Figyeljük meg, hogy a BASIC szimpla és dupla pontosságú formátumaitól eltérően, a REAL4 és a REAL8 kitevőik dinamikus tartományában és a mantissza pontosságában különböznek.

A mantisszamező bináris tört formátumú, az első magasértékű bitet beleértve (mint a régi real-formátumban és a BASIC-nél). A logikai tizedes (vagy kettes) pont a mantissza beleértett magasértékű bitjétől jobbra van. A REAL4-ben a mantissza bitek száma 23, plusz a beleértett magasértékű bit; a REAL8-ban 52 mantisszabit van (plusz a beleértett bit).

A REAL4 részei 32 bitesek (négybájtosak); a REAL8-é 64 bitre vagy 8 bájtra adódnak. Figyeljük meg, hogy a régi realtól és a BASIC-formátumoktól eltérően a REAL4 és a REAL8 részei nem illeszkednek jól a bájtokba! E két formátum nyers diagramja a következő:

REAL4	REAL8
M3 M2 E2M1 SE1	M7 M6 M5 M4 M3 M2 E2M1 SE1

A mantisszabájtok M2-től M7-ig tiszta mantisszabitek. A többi bájt mezők és részek keverékét tartalmazza. Az SE1-ként címkézett bájtok mindkét formátumban az előjelet és a kitevő első 7 bitjét tartalmazzák a következők szerint:

Bit 7	6	5	4	3	2	1	0	Megnevezés
S	.	.	.	.	.	.	.	Előjelbit
.	E	E	E	E	E	E	E	Magasértékű kitevőbitek

Az E2M1-gyel jelölt bájtok tartalmazzák a kitevőbitek maradékát (a REAL4-nél 1 bit, a REAL8-nál 4 bit), és a mantissza első 7 vagy 4 bitjét az alábbiak szerint:

REAL4 Bit 7	6	5	4	3	2	1	0	REAL8 Bit 7	6	5	4	3	2	1	0	Megnevezés
E	.	.	.	.	.	.	.	E	E	E	E	.	.	.	.	Alacsony értékű kitevőbitek
	M	M	M	M	M	M	M	.	.	.	.	M	M	M	M	Mantisszabitek

Bár a két új real-formátum és a régi real-formátum elvileg ugyanúgy működik, mégis szinte minden részlete különbözik: az alkotórészek sorrendje és mérete, a kitevő eltolásjelzése és a bájthatárok.

Korábban ebben a fejezetben a BASIC szimpla pontosságú lebegőpontos formátuma (amely ugyanaz, mint a Pascal egyes változatának régi real-formátuma) ismertetésekor bemutattunk egy programot, amely dekódolja egy szám bitjeit, és meghatározza az értékét. Ugyanezt a gyakorlatot végre lehet hajtani a Pascal 1 változatban a régi real, vagy a Pascal 2 változatban a REAL4 és REAL8 formátumok interpretálására. Az elvek mind a három formátumban ugyanazok, bár a részletek lényegesen öszetettebbek a REAL4 és a REAL8 formátumban mivel a mezők nem illeszkednek olyan jól a bájthatárokba.

Kompatibilitási okokból a fordító 1. változatának van a BASIC-kel kompatibilis lebegőpontos formátuma, és a 2. változatnak vannak formátumai, amelyek kompatibilisek a legtöbb más nyelvvel, beleértve az IBM/Microsoft Fortrant, a Lattice/Microsoft C-t és a 8087 és a 80287 matematikai koprocesszorokat is.

A Pascal assembly interfésze

A következőkben a Pascal-paraméter használatához szükséges fontos részleteket fogjuk tárgyalni. (☛ A bemutatott Pascal példából fogunk kölcsönözni, lecsupaszítjuk azt az alapelemeire.)

```
1    INTERFACE SEGMENT 'CODE'
2              PUBLIC    SEGREAD
3    SEGREAD   PROC      FAR
4              PUSH      BP
5              MOV       BP,SP
6              . . .
7              POP       BP
8              RET       10
9    SEGREAD   ENDP
10   INTERFACE ENDS
11             END
```

Az 1. sorban az INTERFACE név önkényes; kötelező viszont a **'CODE'** osztály a Pascal fordítóval való összeszerkesztéshez (aposztrófok között).

A 2. sorban bármely rutin nevét (pl. a SEGREAD-et) **PUBLIC**-ká kell deklarálni, hogy a szerkesztő hozzákapcsolhassa azokhoz a rutinokhoz, amelyek használják. A 3. sorban a **PROC** (eljárás)-t deklarálni kell **FAR**-ral, mivel a Pascal a külső rutinokat **FAR** típusú hívásokkal hajtja végre.

A 4., 5., 7. és 8. sorok szabályos belépési és kilépési utasítások, amelyek a vermet tartják karban. A 8. sorban 10-es érték képviseli azt a veremméretet, amelyet a paraméterek használnak. Ezzel a tárgykörrel a következő részben fogunk foglalkozni.

A PC-családra vonatkozó szabványos hívásokban a szubrutin paramétereit ilyen vagy olyan formában a verembe helyezzük. A paraméterek verembe helyezésére alapvetően két lehetőség van: vagy a paraméterértékeket helyezhetjük a verembe, vagy egy mutatót, amely a memóriában levő értékre mutat. Mindkét megközelítésnek vannak előnyei; az első könnyebb hozzáférést tesz lehetővé az értékekhez; a második lehetővé teszi, hogy az alprogram a paraméterváltozók módosításával adatokat adjon vissza a hívónak.

A Pascal mindkét módszert használja, és korlátain belül lehetővé teszi a választást. (Ezzel szemben a BASIC csak a második módszert használja, a paramétercímet helyezve a verembe.)

Ha a szubrutin deklarálásakor egy paramétert VAR- vagy VARS-ként határozunk meg, akkor utasítjuk a Pascalt, hogy annak a paraméternek a címét helyezze a verembe. (A Pascal szempontjából a VAR vagy a VARS használata az alprogram számára lehetővé teszi egy változó értékének megváltoztatását; a gyakorlatban azt jelenti, hogy az alprogramnak egy változó címét adjuk.) Ha VAR-t specifikálunk, akkor a cím (adatszegmens eltolásként) egyszerű 2 bájtos szóként adódik át. Ha VARS-t specifikálunk, a cím teljes szegmentált címként (szegmensbekezdéssel és -eltolással), két 2 bájtos szót elfoglalva adódik át. A különbség igen fontos, mivel a RETURN-utasításban a veremből törlendő paraméterbájtok számát is befolyásolja. Ha nem specifikáljuk a VAR-t, a Pascal védeni fogja a paraméter értékét, a szubrutin azt nem tudja módosítani. Ezt két módon lehet elérni, és lényeges, hogy melyiket használjuk. Ha az érték a megfelelő típusú és a verembe helyezhető (pl. integer), akkor az érték maga helyeződik a verembe; egyébként a memóriában egy biztonságos másolata készül, és a másolat címe kerül a verembe. Két fontos különbség van. Az egyik: a hívott rutin a paraméterhez másként fér hozzá, vagy az értéket veszi a veremből, vagy azt a címet, ahol az érték található; a másik: a hívóprogram az értéket a veremben levő címen keresztül kevésbé hatékonyan éri el.

Amikor assembly nyelvű interfészrutint írunk, fontos tudni, melyik módszert használjuk. Ez kitalálható a következő egyszerű módszer használatával. Vegyük a rendelkezésre álló információt, és józan ésszel próbáljuk kitalálni, mi történhet egy adott paraméterkészletben. Teszteket vagy próbaprogramokat futtathatunk, vagy ellenőrizhetjük a fordítóprogram pszeudo-assembly kódlistáját.

Ha bizonytalanok vagyunk, az összes paramétert VAR-ként deklarálva, azok egyszerű címeken adódnak át. A VAR használatának előnye, hogy egyszerű paraméter kezelési minta. Egyetlen hátránya van: egyszerű változóknál két gépi nyelvű utasítással hosszabb mint a VARS – ez azonban kis hátrány.

Egy alprogram a Pascalban értéket adhat vissza, ha FUNCTION (függvény)-ként deklaráltuk. Ha az alprogram elég egyszerű értéket ad vissza, akkor a hívó Pascal-rutin a visszaadott értéket az *AX*-regiszterben várja. Egyébként a memóriában tárolódik azon az eltolási címen, amelyet a függvényparaméterek után helyezünk a verembe. Valójában a Pascal egy kiegészítő VAR típusú paramétert teremt a függvényhez, és a függvényértéket abban a paraméterben adja vissza. A paraméterek vermen keresztüli mozgatására nézzük a következő példát! Tegyük fel, hogy egy assembly nyelvű interfészrutin deklarációja Pascalban a következő:

```
PROCEDURE EXAMPLE (I : INTEGER; VAR J : INTEGER); EXTERNAL;
```

Az assembly nyelvű szubrutin azt találná, hogy az első paraméter – az I – a veremben a [*BP* + 8] eltolásnál volt. Az érték *AX*-regiszterbe való helyezésére az utasítást assembly nyelven a következőképpen kódolnánk:

```
MOV  AX, [BP+8]
```

A második paraméternek – a J-nek – a címe a veremben a [*BP* + 6] eltolásnál található, mivel ez a következő paraméter. Ennek az értéknek az *AX*-regiszterbe helyezése két lépést igényel: a címet lekérdezni a veremből, és a J értékét átmozgatni a következőképpen:

```
MOV    BX, [BP+6]
MOV    AX, [BX]
```

A J értékének módosítása megfordítaná a folyamatot:

```
MOV    BX, [BP+6]
MOV    [BX], AX
```

Az I értékét az alprogramunk nem tudja módosítani, mivel nem tudjuk, hol van az I; csak azt tudjuk, hogy értékének egy másolatát a verembe helyeztük.

Ezek az elvek, ha párosulnak némi józan ésszel és a helyzet jellemző sajátosságainak figyelembevételével, elegendők a Pascal-programok assembly nyelvű interfészeinek sikeres megírásához.

A C nyelv

A C nyelv ismertetésekor a Lattice/Microsoft C fordító 1.04 változatára és egyéb rokon fordítókra (amelyek némelyike Lattice vagy Lifeboat néven ismertek) összpontosítunk.Ugyanúgy, mint más nyelveknél, a konkrét információk nemcsak erre a fordítókra vonatkoznak, hanem érvényesek a többé-kevésbé (vagy teljesen eltérő) fordítókra is.

Érdemes megjegyeznünk, hogy a Lattice/Microsoft C fordító egy kis memóriamodellnek nevezett valamit használ, ami a programkódok és az adatok területét 64 kbájtos szegmensekre korlátozza. Ezzel ellentétben a közepes memóriamodell a programkódok számára olyan címeket használ, amelyek bármilyen méretűt megengednek, míg az adatokat 64K-ra korlátozzák; és a nagy memóriamodell szegmentált címeket használ a kódra és az adatokra (amely lehetővé teszi mindkettő számára, hogy tetszőleges méretűre növekedjenek). A fordítós BASIC és Pascal (amelyeket korábban tárgyaltunk) mind a közepes modellt használja (bár a Pascalban korlátozott mértékben az adatokra vonatkozó szegmentált címek használhatók, tehát így bizonyos szempontból nagy modellű). A Lattice C fordító későbbi változatai (az általunk tárgyalt fordító közeli rokonai) használhatják a fenti három memóriamodell bármelyikét, választásunk szerint. Az egy fordítón belüli három memóriamodell megnehezíti az interfészek tárgyalását, ezért inkább a kis modellre összpontosítunk.

A kis modell használata korlátozza a programkódok és adatok méretét, de ez a leghatékonyabb formátum. Látni fogjuk a kis modell hatásait az adatcímformátumban. (☛ L. a következőkben a szubrutin belépéssel és kilépéssel, valamint a paraméter-hozzáféréssel kapcsolatos fejtegetésünket.)

A C nyelv adatformátumai

A C nyelvben három fő adatformátum van: integer, lebegőpontos és string. A Lattice/Microsoft fordítónak három külön integerformátuma és két lebegőpontos formátuma van.

Az integer adatformátumok

A négy integerformátum a következő: a CHAR, az INT, az UNSIGNED és a LONG. A CHAR-formátum 1 bájtot foglal el, és olyan előjel nélküli integerként kezeljük, melynek értéke 0-tól 255-ig terjed. Az INT-formátum 2 bájtot foglal el, és előjeles értékként kezeljük úgy, hogy −32 768-tól +32 767-ig terjedhet az értéktartománya. Az UNSIGNED-formátum (amelyet a szokásos C-szóhasználattal UNSIGNED INT-ként is ismerünk) 2 bájtot foglal el. Előjel nélküli értékként kezeljük, és értéktartománya 0-tól a 65 535-ig terjed. A LONG-formátum 4 bájtot foglal el, és előjeles értékként kezeljük, így az értéktartománya −2 147 483 648-tól +2 147 483 547-ig terjed.

Bár ezek az integerformátumok viszonylag egyszerűnek tűnnek, néhány dolgot máris érdemes megbeszélni. Meg kell jegyeznünk, hogy a C nyelv általános definíciója megengedi a

legtöbb alapvető adattípus LONG (hosszú) és SHORT (rövid), ill. előjeles és UNSIGNED (előjel nélküli) változatait. Ennél a fordítónál a SHORT INT egy 2 bájtos, előjeles érték (ugyanaz, mint az INT), és nincs sem a 4 bájtos LONG-nak előjel nélküli (UNSIGNED), sem az 1 bájtos CHAR-nak előjeles változata. A címértékek az adatszegmenshez viszonyított eltolás formájában jelennek meg, és a 16 bites UNSIGNED INT-formátumban tároljuk.

A C nyelvnek általában egy kicsit kusza a karakterek és a számok közti kapcsolata, különösen a CHAR és az INT közötti hasonlóságokat és különbségeket nézve. A fordítónál a CHAR-ok szigorúan 1 bájtot elfoglaló számok, amelyek értéke 0-tól 255-ig terjed. A CHAR-okat azonban kifejezhetjük karakteralakban is. A Q karakter pl. 81 számértékre fordítódik; de ha a C-programban 81-et írnánk, azt integertípusnak tekintené, nem pedig egy 1 bájtos CHAR-nak (hacsak nem tettünk valamit azért, hogy CHAR-formátumba kényszerítsük). Mivel a C, eltérően a Pascaltól, nem tesz határozott különbséget karakterek és számok között, integerformátumok (INT vagy UNSIGNED) és karakterformátumok (CHAR vagy stringelemek) között tudjuk az értékeket konvertálni. (A szabványos Pascalnak része a **CHR** és az **ORD** függvény, amelyek az egész számokat és a karaktereket alakítják át egymásba. A lektor.)

A string adatformátumok

A C nyelv stringkezelése tükrözi a karakterek kettős – vagy karakterkénti, vagy számkénti – kezelését. A C-ben egy karakterstring a CHAR-ok olyan sora, amelyet egy zérókarakter követ, ez a string végét jelzi. A stringet lezáró zérókarakter miatt a string bármilyen hosszú lehet, és nincs egy kifejezett hosszúságjelző (mint a Pascalban vagy a BASIC-ben), amely be tudná állítani a maximális méretet. Mivel nincs hosszúságjelző, a stringet végig kell vizsgálni, hogy megkapjuk egy string hosszát. A string zérókarakterrel fejeződik be, a zérókarakter (**CHR$(0)** a BASIC jelölésben) viszont nem lehet része egy stringnek, ami nem nagy veszteség. A lezáró zérókarakter miatt nincs különbség a fix és a változtatható hosszúságú stringek között.

A C-fordító magára vállalja a stringet záró bájt hozzáadásának feladatát, nem kell odatennünk. Ha pl. egy programban az *ABC*-t írjuk, a C-fordító egy 4 bájtos stringet alkot, amely az általunk írt 3 bájtos és az azt követő zéróbájtot tartalmazza. Figyelem: bár a C-fordító bármely stringkonstanshoz hozzáadja a záróbájtot, és a szabványos C stringkezelő szubrutinjai (szükség szerint) foglalkoznak a zéróbájttal, a zéróbájt nem magától jelenik meg! Ha saját stringkezelő programokat írunk (ami elég általános), a zéróbájtra figyelnünk kell.

Lebegőpontos adatformátumok

Ez a fordító két lebegőpontos formátumot használ: FLOAT és LONG FLOAT (vagy DOUBLE). A FLOAT-formátum négybájtos, a LONG FLOAT-formátum pedig nyolcbájtos. Követik a 8087 és a 80287 matematikai koprocesszorok és más programozási nyelvek által használt szabványos formátumokat. Mi csak vázoljuk a két formátumot. (☛ A REAL4 és a REAL8 ismertetésekor már részleteztük.)

A két lebegőpontos formátumot úgy a legkönnyebb megérteni, ha bitek sorozatának tekintjük azokat (32 bit a FLOAT-nál, 64 bit a LONG FLOAT-nál), amelyeket „fordított szavakként" tárolunk, vagyis a legnagyobb helyiértékű biteket a legutolsó bájton. A lebegőpontos formátumokat bitsorrendben, a legnagyobb értékű bittől indulva írjuk le.

Első helyen áll a legnagyobb értékű bit, az előjelbit: 0, ha pozitív, 1 ha negatív. A következő bitcsoport (8 a FLOAT-nál, 11 a LONG FLOAT-nál) határozza meg a bináris kitevőt,

eltolásos formátummal. Az eltolás 127 a FLOAT-nál és 1 023 a LONG FLOAT-nál. Ha vesszük a kitevőbitek előjel nélküli bináris egész értékét, és levonjuk az eltolás értékét (127 vagy 1 023), megkapjuk az igazi kitevőértéket. A megmaradó bitek (23 vagy 52), a mantissza vagy a törtrész. A beleértett magasértékű 1. bit a többi mantisszabit előtt áll. (☞ A Pascal nyelvnél már ismertetett táblázat mutatja az előjel-, a kitevő- és a mantisszabitek elhelyezkedését a két formátum 4 vagy 8 bájtjában.)

A C nyelv asembly interfésze

Az interfész programok általános ötszintes megközelítése éppúgy vonatkozik a C-fordítókra, mint az előzőkre. A C-fordítóval használható interfész alapszerkezetének bemutatására az előző fejezetből veszünk kölcsön egy programot, és lecsupaszítjuk azt a legfontosabb elemeire. Az eredeti programot arra terveztük, hogy a DOS dátum- és időszolgáltatásait használva, a hét napjait számolja ki. (☞ L. a 16. fejezet végén.)

```
 1           PGROUP     GROUP PROG
 2  PROG     SEGMENT    BYTE PUBLIC 'PROG'
 3           PUBLIC     WEEKDAY
 4           ASSUME     CS:PROG
 5  WEEKDAY  PROC       NEAR
 6           PUCH       BP
 7           MOV        BP,SP
 8           . . .
 9           POP        BP
10           RET
11  WEEKDAY  ENDP
12  PROG     ENDS
13           END
```

Az 1. sor segít az assembly nyelvű interfészt a C-fordító által használt szerkesztőkonvenciókkal való koordinálásában.

A 2. sorban a **PROG** név és a **'CODE'** osztályozás (aposztrófok között) is azért szükséges, hogy kielégítsük a szerkesztés elvárásait.

A 3. sorban bármely rutinnevet (pl. a WEEKDAY-t,) **PUBLIC**-ká kell deklarálnunk azért, hogy a szerkesztő hozzászerkessze azokhoz a rutinokhoz, amelyek azt használják.

A 4. sorban az **ASSUME** azért kell, hogy összeállítsa ezt a **NEAR**-eljárást akkor is, ha nem teszünk semmilyen programcím-referenciát.

Az 5. sorban a **PROC NEAR**-t kell deklarálni, mivel ez a C-fordító külső rutinjaihoz **NEAR** típusú hívásokat végez. Ez a jellemző eltér a legtöbb nyelvtől, és az első, a három fontos eltérés közül.

A 6., a 7., a 9. és a 10. sorok képviselik a szabványos belépő és kilépő kódokat, amelyek a vermet kezelik.

A 10. sorban a **RET**-utasítást nem követi érték. A szubrutin paramétereit a hívó helyezi a verembe, és vagy a hívott szubrutin, vagy a hívó (miután a szubrutin visszaadja a vezérlést) távolítja el a veremből. Mint láttuk, a legtöbb programozási nyelv ezt a feladatot a hívott szubrutinnal hajtatja végre; mi határozzuk meg a **RET**-utasításban az eltávolítandó paraméterbájtok számát. Ez a C-fordító a paraméterek veremből való törlésére rendelkezik egy hívó szubrutinnal is, így **RET**-ünket nem követi szám. Ez a második a három közül, amelyben a C eltér az interfész szokásos szabályaitól.

Vegyük észre, hogy a C megengedi a változó számú paramétereket: ez szabványos része a C nyelvnek, pl. a sokat használt **PRINTF**-rutinnál van rá szükség. A hívott rutin a veremmutatók és a *BP*-értékek ellenőrzése révén meghatározhatja, hogy mennyi paraméter került a verembe.

A 11–13. sorokat az interfészrutin befejezésére használtuk.

Paraméterátadás a C nyelvben

A paramétereket a verembe közvetlenül vagy közvetve helyezzük. A C-fordító a paramétereket a verembe az írásmódhoz képest fordított sorrendben helyezi el, tehát az első paraméter kerül legutoljára a verembe, és így a legtetejére: azaz ennek van a legkisebb eltolása a *BP*-regiszterhez képest. Ez a legtöbb nyelvben használt konvencióknak az ellentéte. A C-interfészekre vonatkozó három rendkívüli dolog közül ez a harmadik.

Mint már említettük, a paramétereket vagy az értékük, vagy a címük verembe helyezésével lehet átadni. Hacsak nem utasítjuk másként, a C-fordító a cím helyett az értéket kísérli meg elhelyezni a veremben. Ez a megállapítás kicsit csalóka lehet, mivel a C sok mindent címnek tekint. Ha pl. van egy *S* nevű stringváltozónk, a C az *S* név értékét egy stringcímnek tekinti, nem pedig egy stringnek. Ha az *S* egy stringváltozó, az *I* pedig egy integer, a szubrutin a következő:

SUBR (S, I)

Erre az *S* és az *I* értékei is a verembe kerülnek: az *I* értéke egy integer, az *S* értéke pedig egy cím (egy stringérték címe).

A C-ben a változónév elé tett & jellel kikényszeríthetjük, hogy egy paraméter címe adódjon át:

SUBR (S, &I)

amely egy *I* változó címét adja át. Ebben a példában a SUBR-szubrutin két címet kapna, mind a kettő egy-egy értékre mutat. Ha a következő hibát követnénk el,

SUBR (&S, &I),

akkor egy string címét adnánk át – feltehetően nem ezt akartuk!

Kezdő C-programozók könnyen zavarba jöhetnek a paraméterátadás során. Szerintünk ez a legnagyobb hibaforrás a C nyelven írt assembly interfészek használatában. Mindenki támaszkodjon a józan eszére, és gondosan elenőrizze a rutinjait, hogy tényleg az történik-e az interfész paraméterekkel, amit gondolt.

Összehasonlítva a Pascalt a C-vel: az & jel használata a subrutin paramétere előtt egyenértékű a paraméter **VAR**-ként való deklarálásával a Pascalban: ennek következtében a verembe egy cím, és nem egy érték kerül.

A paraméterekhez a veremből férünk hozzá, és a paraméterek adatait a szokásos módon módosítjuk a paramétercímek verembe helyezésével. A C-interfészekre vonatkozó legfőbb különlegesség: a veremeltolások eltérőek azoktól, amelyekkel rendszerint találkozunk. Mivel a szubrutinokat a **FAR CALL** helyett **NEAR CALL**-lal hívjuk, a veremben levő legközelebbi paraméter a [*BP*+4] eltolásnál lesz a szokásosabb [*BP*+6] eltolás helyett – ahogy már ezt az előzőekben említettük –; a legközelebbi paraméter nem az első, hanem a legutoljára beírt lesz.

A paraméterek hozzáférésének és módosításának konkrét példájaként figyeljünk meg egy szubrutint, amelyet két integerrel a következőképpen hívunk:

SUBR (&I, J)

és amelynek az a feladata, hogy a második paraméterből az értéket az elsőbe helyezze (a szubrutint egyenlővé téve az I = J értékadó utasítással). A következő:

```
MOV    AX,[BP+6]    ;a J érték lekérdezése
MOV    BX,[BP+4]    ;az I cím lekérdezése
MOV    [BX],AX      ;a J érték I-be helyezése
```

A C nyelvben az összes szubrutin általában integer típusú értéket ad vissza, még akkor is, ha a visszaadott értéknek számunkra nincs jelentése. Az értéket a hívó használhatja, vagy figyelmen kívül hagyhatja. (A C ismét különösen viselkedik – a legtöbb más programnyelv világosan megkülönbözteti azokat a szubrutinokat, amelyek visszaadnak értéket, és azokat, amelyek nem.) Az általános konvenciót követve, a C ezt a visszaadott értéket az *AX*-regiszterben várja. Ha van egy visszaadandó értékünk, azt az *AX*-be helyezzük; ha nem, és aprólékosabbak akarunk lenni, akkor mielőtt viszatérnénk, az *AX*-et zéróra állítsuk be, így meggyőződhetünk arról, hogy következetes, nem véletlenszerű értéket adunk vissza.

Zárómegjegyzés

Bár csak néhány programozási nyelv rövid analízisét mutattuk be, azt reméljük, talán ennyi is segített azok jellemzőinek megértésében. Jó volna persze a PC-családoknál használt minden egyes nyelv jelemzőinek összehasonlítása, de ez lehetetlen egy ilyen természetű könyvben. Az öt programozási nyelv öszehasonlításának szempontjai azonban segíthetnek más nyelvek vizsgálatakor is.

A MELLÉKLET
Installálható berendezésmeghajtók

Bevezetés • 342

Általános áttekintés • 342

Az ANSI-meghajtó • 343

Az ANSI-képernyővezérlő • 344
Az ANSI-billentyűzetvezérlő • 344
Az ANSI-meghajtó előnyei és hátrányai • 344

Bevezetés

Az a két jelemző, amely a DOS 2.00 változatban jelent meg, külön fejezetet érdemel: az installálható berendezésmeghajtók általában és az ANSI-meghajtó (amelyet ANSI.SYS-nek is hívunk). Ezek kapcsolódnak egymáshoz: egyrészt mindkettő a DOS 2.00 újdonsága, másrészt az ANSI-meghajtó maga is egy installálható meghajtó –, de programozási perspektívából egymástól gyökeresen különbözőek. A berendezésmeghajtók vizsgálatával kezdjük, majd áttérünk az ANSI-meghajtó részletesebb ismertetésére.

Általános áttekintés

A DOS szerves része a számítógép legáltalánosabb berendezéseivel (lemezegységek, kommunikációs vonalak, nyomtatók, billentyűzet és monitor) való együttműködés. Még számos más fajtájú berendezést is lehet a számítógéphez csatlakoztatni. Az összes ilyenfajta csatlakoztatás általában valamilyen szoftvertámogatást is kíván, amelyet berendezésmeghajtóknak (device driver) nevezünk, és amelyek a berendezéseket a DOS-hoz és a DOS-programokhoz kötik.

A 2.00 változat kiadása óta a DOS képes bármilyen berendezésmeghajtót befogadni, amely a szabványos integrálási szabályoknak eleget tesz. A berendezésmeghajtó programot a lemezről lehet beolvasni, és a DOS a bekapcsolási eljárásban építi magába.

Egy CONFIG.SYS nevű lemezállomány közli a DOS-szal, ha van betöltendő berendezésmeghajtó. A berendezésmeghajtó nevét és az állomány helyét a CONFIG.SYS állományban levő állományspecifikáció (**DEVICE**) utasítássora azonosítja. Minden egyes ilyen utasításra a DOS megkeresi a programot, betölti azt a memóriába, és végigmegy az összes olyan szükséges lépésen, ami a berendezésmeghajtót a DOS környezetébe beépíti. Jellemző a lemezmeghajtó, amely egy régóta meglévő eszközt újszerűen kezel. Vegyük pl. az új lemezmeghajtó hardvert, amelynek parancsai újak ugyan a DOS számára, de általános jellemzői megegyeznek a többi hasonló hardverével. Ilyenkor a meghajtó az elődei által meghatározott formátumot követi. A hardver tehát új, de a DOS ugyanúgy használja, mint a régit. Hasonlóképpen egy olyan berendezésmeghajtó, amelyik egy kiegészítő egeret (mause) vagy botkormányt (joystick) kezel, azokat billentyűzetszerű készülékekként kezelheti. A berendezésmeghajtók azonban végezhetnek olyan funkciókat is, amelynek kevés (vagy semmi) köze sincs a számítógépet kiegészítő új hardverhez; ilyen az ANSI-berendezésmeghajtó, amelyet a következő részben tárgyalunk. Az ANSI-berendezésmeghajtó nem bővíti új hardverrel a számítógépet, hanem módosítja a számítógép szabványos hardverének (a billentyűzetnek és a monitornak) a működését.

A berendezésmeghajtó írásának műszaki részletei valójában egy DOS rendszerprogramozási könyvébe tartoznak, de a legfőbb szempontokat megadjuk itt. Magának a berendezésmeghajtó-állománynak egy közel szabványos programformátuma van, a meghajtóra jellemző néhány azonosító információval kiegészítve. Kétfajta berendezésmeghajtó van: a karakterberendezések, amelyek a billentyűzethez, a képernyőhöz, a nyomtatóhoz vagy a kommunikációs porthoz hasonlóan a karakterek soros áramlásával működnek; és a tömbkészülékek, amelyek (mint a lemezegység), valamilyen formájú tömbcím révén azonosított random adattömböket írnak és olvasnak. A karakterberendezéseket a saját nevükkel azonosítjuk, hasonló nevekkel, mint az LPT1: vagy a COM1:, és azokat állományként kezelhetjük. A tömbkészülékeket azzal a lemezegységbetűvel azonosítjuk, amelyet a DOS jelöl ki, és amelyek azonosak a lemezegység azonosító betűivel *A, B, C* stb.

A berendezésmeghajtó program állományának a DOS számára számos belépési pont címét kell megadnia, ahol a meghajtó különböző célokra meghívható: inicializálásra, az igények

kiszolgálására, nagyobb munkák végrehajtására. A meghajtóprogramnak értenie kell a DOS szabványos utasításkészletét, amellyel az eszközökkel kommunikál, pl. felszólítja azokat valamilyen parancs végrehajtására, vagy lekérdezi a készülék állapotát.

A berendezésmeghajtó írása rokon a BIOS-programok írásával, a DOS-hoz és a beépített ROM-BIOS-hoz is közel állnak. Ez ugyanakkor a legfejlettebb és a legbonyolultabb programozási feladatok közé tartozik, amire valaki valaha is vállalkozott.

Az ANSI-meghajtó

A DOS-hoz installálható berendezésmeghajtóknak egyike az ANSI-meghajtó. Egy olyan program, amely nagyban kiterjeszti a billentyűzetbemenet és a monitorkimenet kezelését. A DOS IBM-változataiban az ANSI csak akkor aktív, ha az indulási művelet során a DOS által betöltendő CONFIG.SYS-állomány révén installáltuk. A CONFIG.SYS-állományban az ANSI-meghajtó aktivitálására használt utasítás a következő:

```
DEVICE = ANSI.SYS
```

Érdemes megjegyezni, hogy míg az ANSI-meghajtó egy választható része a IBM-DOS változatainak, addig kötelezően része néhány ahhoz hasonló számítógépen használt DOS-nak (amelyek azonban nem teljesen kompatibilisek az IBM PC-családdal). A Texas Instruments Professional számítógépeinek (általánosan TI Pro-néven ismert) és a NEC Advenced Personal Computer III-jának is (NEC APC–III) része a beépített ANSI-meghajtó a DOS 2 változataiban.

Az ANSI-meghajtó kijelzi mindazokat a képernyőkimeneteket és billentyűzetbemeneteket, amelyek a szabványos DOS képernyő- és billentyűzetszolgáltatásokon keresztül kerülnek átadásra. (Fontos megjegyezni, hogy bármely olyan billentyűzet- vagy képernyőadatot, amely elkerüli a DOS-t az ANSI-meghajtó sohasem ismeri, és nem dolgozza fel.)

A képernyőkimenet figyelése során az ANSI-meghajtó olyan kódokra vadászik, amelyek a meghajtóra vonatkozó utasításokat azonosítják. A meghajtó figyelembe vesz bármit, amit utasításként ismer fel, és azután azt eltávolítja, így az utasításkód nem jelenik meg a képernyőn. Valójában a meghajtó ebből a szempontból egy kétállású kapcsolóként működik: figyeli a képernyő felé irányuló kimenetet, és átenged bármit, ami nem a meghajtónak szóló utasítás; azonban minden meghajtóutasítást az utasítás feldolgozó részébe irányít.

Az ANSI-meghajtó utasításait egy 2 bájtos kód azonosítja: az első bájt *escape*-karakter (hexa 1B vagy **CHR$(27)**), a második egy kezdő szögletes zárójel ([) (hexa 5B vagy **CHR$(91)**). Ezeket az azonosító bájtokat követik az utasításparaméterek, és végezetül maga az utasításkód. Az utasításparaméterek vagy számok (decimális számjegyként értelmezett ASCII-kódba levő számkarakterek), vagy „zárt ASCII-karakterstringek". Ha egynél több paraméter van, azokat pontosvesszővel választjuk el. Az ANSI-meghajtó utasítását befejező utasításkód maga mindig egy egyszeri alfabetikus karakter. Az utasításban különbözőnek számít a kisbetű és a nagybetű: pl. a *h* és a *H* két különböző utasítás.

Két példát mutatunk egy egyszerű és egy összetett esetre (a csillag jelöli az *escape*-karaktert, hexa 1B):

```
* [1C
* [65;32;66;"Re–mapped B"p
```

Az ANSI-meghajtó sok utasítást felismer, ezek két nagyobb kategóriába tartoznak: képernyővezérlő vagy billentyűzetfordító utasítások lehetnek. Nézzük meg először a képernyő vezérlését!

Az ANSI-képernyővezérlő

Bár a PC-re vonatkozó BIOS-szolgáltatások megengedik, hogy a kurzort bárhova vigyük a képernyőn, és bármit csináljunk, ami nekünk teljes képernyővezérlést ad; a szabványos DOS szolgáltatásai ezt nem nyújtják. Valójában a DOS képernyőkimenet-szolgáltatások teljes egészében TTY (vagy teletype) kimenet-beállítottságúak: olyan kimenetek, amelyek nem tudnak többet egy nyomtatónál. Ez természetesen figyelmen kívül hagyja a képernyő gazdagabb lehetőségeit. A DOS-ban a képernyőkimenet hiányosságai késztetik a legtöbb programot arra, hogy a DOS-szolgáltatásokat megkerülje, és alacsonyabb (BIOS) szolgáltatásokat használjon.

Az ANSI-meghajtó képernyővezérlő utasításai a teljes képernyőt kezelő utasításkészlettel orvosolják ezt a helyzetet, csaknem mindenre felkészültek, amire egy képernyő képes. Az utasítások elvégzik a kurzor mozgatását, a képernyőtörlést, a képernyő-attribútumok (szín, aláhúzás, villogás stb.) beállítását, és a grafikus és a szöveges mód közötti oda-vissza váltást. A kifinomultság további szintjeként vannak olyan utasítások, amelyek az aktuális kurzorhelyet kimentik, így a kurzor elmozgatható valamilyen információ kijelzésére, és utána visszatérhet az eredeti helyére.

Az ANSI-billentyűzetvezérlő

Az ANSI-meghajtó által elfogadott másféle utasítás a billentyűzetfordító utasítás. Ha ezen utasítások egyikét adjuk a meghajtónak, az kijelzi a bemenetet, és a billentyűzetkaraktert egy másik karakterrel vagy karakterstringgel helyettesíti. Ezért az ANSI-meghajtó a népszerű billentyűzetkiegészítő programoknak (pl. a ProKey) nyers, de hatékony helyettesítője lehet.

Bár az ANSI-meghajtónak ez a két típusú utasítása szándékukat és felhasználásukat illetően nagyon különböző, a meghajtóhoz hasonló módon kerülnek: a képernyőre kimenő karakterek soraként.

Az ANSI-meghajtó előnyei és hátrányai

Az ANSI-meghajtó utasításaira kétféleképpen lehet nézni: a felhasználó szemszögéből (aki az ANSI-meghajtót néhány előnyös trükk végrehajtására használhatja) és a programozó szempontjából (aki a programfejlesztéshez használhatja segédletként). Ez egy programozói könyv, de röviden megemlítünk az ANSI-meghajtóval kapcsolatos néhány általános felhasználói trükköt.

A felhasználók leggyakrabban az ANSI-meghajtót „a szegény ember ProKey-jének" és a DOS egy promptbővítőjének tekintik. A billentyűzetfordító utasítások használatával lehetséges a ProKey-program durva szimulálása. Általában a billentyűzetutasításokat, azok szövegállományba helyezésével és (TYPE-utasítás útján) képernyőre (így az ANSI-meghajtóhoz) küldésével aktiváljuk. Az ANSI-meghajtó utasításainak a prompt stringbeágyazásával a DOS utasításpromptját negyedik dimenzióba tegyük. Egy ilyen ötletes prompt a kurzort a képernyő tetejére viheti, és az a dátumot és az időt inverz megjelenítéssel jelezheti. Utána a kurzort visszaadhatja a rendes helyére, vagy még igényesebb megoldásként törölheti a képernyőt, hogy utána egy teljes menüt jeleníthessen meg. A lehetőségek végtelenek.

A programozó szempontjából az ANSI-meghajtó merőben más. Használata a programozónak két lényeges előnyt nyújt, mindkettő igen fontos lehet. Akiknek nincs meg a tudásuk vagy eszközük arra, hogy a szükséges assembly nyelvű interfészeiket a BIOS-szolgáltatásokhoz

megépítsék; az ANSI-meghajtó bármely programozási nyelvben lehetővé teszi a leglényegesebb BIOS típusú szolgáltatásokat. Ez nagyon előnyös lehet azoknak is, akik nem a PC-családhoz kötődő programokat akarnak írni, hanem az ANSI-meghajtót használó számítógépen működőket.

E nyilvánvaló előnyök ellenére az ANSI-meghajtó utasításainak programjainkban való használata általában nem túl szerencsés ötlet. Egyfelől azt kívánja, hogy legyen ANSI-meghajtó bármilyen számítógépben, amelyen a programjainkat használjuk. Ez jelentősen növeli azokat az utasításokat, amelyeket a program környezetének kialakításához ki kell adnunk. Elég nehéz elmagyarázni programunk működési környezetét és használatát az új felhasználóknak és a szakembereknek anélkül, hogy olyan különleges bonyolult feladatot (mint az ANSI-meghajtó installálása) is elmagyaráznánk.

Az ANSI-meghajtó következő hátránya, hogy nem mindig használható. Az IBM ablakrendszer, a Topview, pl. nem nyújtja az ANSI-meghajtó jellemzőit, tehát azok a programok, amelyek igénylik a meghajtót, nem használhatják a Topview-t. Ez igaz lehet más környezetekre is.

A legfontosabb tény, hogy összehasonlítva más rendelkezésre álló módszerekkel, az ANSI-meghajtó a teljes képernyőkimenet generálásában szánalmasan lassú. Az ANSI-meghajtó, a PC BIOS-szolgáltatások és a képernyőmemória közvetlen írásának relatív sebessége összehasonlítható a Norton Utilities 3. változatának NU-programját próbálgatva. A NU-program három képernyőmeghajtója a három kimenetkezelési módszert használja. Mindegyiket kipróbálva hamar meg lehet győződni arról, mennyivel lassabb az ANSI-meghajtó. Hacsak nem igen kevés a kimenet, az ANSI-meghajtó túl lassú.

B MELLÉKLET
Hexadecimális aritmetika

Bevezetés • 348

Bitek és a hexadecimális értékek • 349

A szegmentált címek és hexadecimális jelölésük • 350

Decimális-hexadecimális konverzió • 353

Hexadecimális aritmetika BASIC nyelven • 354

Hexadecimális összeadás • 354

Hexadecimális szorzás • 354

Bevezetés

A hexadecimális számok azért bukkannak fel, mert a hexadecimális (16-os számrendszerbeli) jelölés nagyon kényelmes módja a bináris (2-es számrendszerbeli) számok írásának.

A hexadecimális számok alapszáma a 16 ugyanúgy, mint a decimális (10-es számrendszerbeli) számoknál 10-es; az a különbség, hogy a hexadecimális számokat 16 szimbólummal, a decimális számokat pedig 10 szimbólummal írjuk le (0–9). (A hexadecimális és a *hexa* kifejezést egymással felcserélhetően használjuk.) A hexa jelölésben a 0-tól 9-ig terjedő tíz jelet használjuk a tíz érték jelölésére, a megmaradó hat értéket (11-től 15-ig) pedig az *A*-tól *F*-ig terjedő betűkkel jelöljük (lásd B1. táblázat).

Az A-tól F-ig terjedő hexa számokat általában nagybetűvel írjuk, de néha találkozhatunk a kisbetűs jelöléssel is: a jelentésük azonban ugyanaz.

A hexa számok felépítése ugyanolyan, mint a 10-es számrendszerbeli számoké; ha pl. a 123-as számot írjuk, akkor a 10-es számrendszerben ezalatt a következőt értjük:

1-szer 100 (10-szer 10)
+2-szer 10
+3-szor 1

Ha a 123-at hexa számként használjuk, akkor a következőt értjük ezalatt:

1-szer 256 (16-szor 16)
+2-szer 16
+3-szor 1

Úgy tűnik, nincs szabványos mód a hexa számok írására, különböző helyeken különbözőképpen találjuk. A BASIC az &H előtagot használja a hexa számok azonosítására, és ezt a jelölést néha másutt is használják. Esetenként a #-t vagy a 16#-t használják, de leggyakrabban a hexa számokat egy H követi. A hexadecimális számok kifejezésére használt legáltalánosabb jelölés a szakirodalomban mégis a mindenféle külön jelölés nélkül álló szám. Az összefüggések alapján kell felismerni, hogy egy szám mikor tízes, és mikor hexadecimális. Ha egy leírásban olyan szám látható, amely decimális számnak tűnik, gondosan ellenőrizni kell: lehet, hogy valójában hexadecimális. Ebben a könyvben a hexadecimális számok azonosítására az eléjük toldott hexa megjegyzést használtuk.

B1. táblázat
A tizenhatos számrendszerben lévő számjegyek tizes értékei

Hexa	Dec	Hexa	Dec
0	Nulla	8	Nyolc
1	Egy	9	Kilenc
2	Kettő	A	Tíz
3	Három	B	Tizenegy
4	Négy	C	Tizenkettő
5	Öt	D	Tizenhárom
6	Hat	E	Tizennégy
7	Hét	F	Tizenöt

Amikor hexa számokkal kell dolgozni, a BASIC használható segédletként (☛ l. később), de manuálisan is feldolgozhatók. Bármely módszernek hasznos kiegészítői lehetnek a függelék végén lévő konverziós és aritmetikai táblák. Mielőtt eljutnánk a táblákhoz, először elmagyarázzuk, hogy miért olyan kompatibilisek a hexadecimális és bináris számok. Ezután leírjuk a programozásban leggyakrabban használt hexa számok felhasználásának egyikét, a szegmentált címzést.

Bitek és a hexadecimális értékek

A tizenhatos alapú számjegyeket elsősorban a kettes számrendszerbeli számoknak a gyorsírásaként alkalmazzuk. Minden hexadecimális számot a bináris információ négy bitje alkotja (l. B2. táblázat). A bináris (2-es alapú) számrendszerben a 4 bites számnak 16 különböző kombinációja lehet, tehát arra, hogy a 4 bites számot egyetlenegy számértékkel mutassuk be, csak egy

B2. táblázat
A tizenhatos számrendszerben lévő számjegyek bitértékei

Hexa	Bitek	Hexa	Bitek
1	0 0 0 0	8	1 0 0 0
1	0 0 0 1	9	1 0 0 1
2	0 0 1 0	A	1 0 1 0
3	0 0 1 1	B	1 0 1 1
4	0 1 0 0	C	1 1 0 0
5	0 1 0 1	D	1 1 0 1
6	0 1 1 0	E	1 1 1 0
7	0 1 1 1	F	1 1 1 1

B3. táblázat
Egy bájtban és egy 2 bájtos szóban levő bitek tizenhatos és tizes számrendszerbeli értékei

Bit	Szó	Bájt	Érték	
			dec	hexa
0	 1	 1	1	1
1	 1 .	 1 .	2	2
2	 1 . .	 1 . .	4	4
3	 1 . . .	 1 . . .	8	8
4	 1	. . . 1	16	10
5	 1	. . 1	32	20
6	 1	. 1	64	40
7	 1	1	128	80
8	 1		256	100
9	 1		512	200
10	 1		1 024	400
11	 1		2 048	800
12	. . . 1		4 096	1000
13	. . 1		8 192	2000
14	. 1		16 384	4000
15	1		32 768	8000

lehetőség van: a 16-os alapú számrendszer használata. Éppen ezért használjuk számítógépeinkkel a hexadecimális aritmetikát (l. B3. táblázat).

Amikor a kétbájtos szavakat használjuk, emlékezzünk a bájtok fordított sorrendjére, arra a sorrendre, ahogy a memóriában ezeket tároljuk. (☛ L. a 3. fejezetet.)

A szegmentált címek és hexadecimális jelölésük

A hexadecimális számok legáltalánosabb felhasználásának egyike a memóriacímzés. A 2. és 3. fejezetből talán már ismert, hogy a teljes cím 20 bit vagy 5 hexadecimális számjegy széles. Mivel a PC 8088-as mikroprocesszora csak 16 bites számokkal tud dolgozni, a címeket le kell bontani két 16 bites szóra, ezeket szegmentált bekezdésnek és relatív eltolásnak nevezzük. A két részt együtt 1234:ABCD-ként írjuk. A szegmentált részt írjuk először, és mindkét részt hexadecimális alakban adjuk meg.

A cím szegmentált részét úgy kezeljük, mintha 16-tal megszoroztuk volna, vagyis egy hexadecimális 0-át írtunk volna utána. A két rész összeadva kiadja a tényleges 20 bites címet.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1		2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2			4	5	6	7	8	9	A	B	C	D	E	F	10	11
3				6	7	8	9	A	B	C	D	E	F	10	11	12
4					8	9	A	B	C	D	E	F	10	11	12	13
5						A	B	C	D	E	F	10	11	12	13	14
6							C	D	E	F	10	11	12	13	14	15
7								E	F	10	11	12	13	14	15	16
8									10	11	12	13	14	15	16	17
9										12	13	14	15	16	17	18
A											14	15	16	17	18	19
B												16	17	18	19	1A
C													18	19	1A	1B
D														1A	1B	1C
E															1C	1D
F																1E

B1. ábra. Két hexadecimális szám összeadása

Az 1234:ABCD szegmentált címet pl. a következőképpen alakíthatjuk címmé:

```
  1 2 3 4 0      (Figyeljük meg a jobb oldalon hozzáadott nullát!)
+   A B C D
-----------
  1 C F 0 D
```

Ha arra van szükség, hogy kiszámoljuk azt a tényleges címet, amelyre a szegmentált cím mutat, ezt a képletet kövessük. (A B1.- B2. ábra is segíthet.)

MEGJEGYZÉS: Vegyük figyelembe, hogy ugyanazt a tényleges címet számos eltérő, különálló szegmentált címmel lehet leírni.

Nincs legjobb módszer a tényleges cím szegmentálására. A legegyszerűbb eljárás az, hogy szegmentált bekezdésrészként a tényleges cím első számértékét vesszük, amelyet három zéró követ, a megmaradó négy számot pedig relatív részként tekintjük. Ezt a szabályt követve az előző 1CF0D a következőképpen bontható fel: 1000:CF0D. Az IBM Technical Reference kézikönyv IBM ROM-BIOS listázása ezt a konvenciót követi úgy, hogy az ott megjelenő relatív címeknek a szegmentált bekezdés része (amit nem mutat) F000.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2			4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3				9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4					10	14	18	1C	20	24	28	2C	30	34	38	3C
5						19	1E	23	28	2D	32	37	3C	41	46	4B
6							24	2A	30	36	3C	42	48	4E	54	5A
7								31	38	3F	46	4D	54	5B	62	69
8									40	48	50	58	60	68	70	78
9										51	5A	63	6C	75	7E	87
A											64	6E	78	82	8C	96
B												79	84	8F	9A	A5
C													90	9C	A8	B4
D														A9	B6	C3
E															C4	D2
F																E1

B2. ábra. Két hexadecimális szám szorzata

B4. táblázat

A hexadecimális számhelyek értékeinek decimális megfelelője

Első pozíció

hexa	dec	hexa	dec
0	0	8	8
1	1	9	9
2	2	A	10
3	3	B	11
4	4	C	12
5	5	D	13
6	6	E	14
7	7	F	15

Második pozíció

hexa	dec	hexa	dec
0	0	8	128
1	16	9	144
2	32	A	160
3	48	B	176
4	64	C	192
5	80	D	208
6	96	E	224
7	112	F	240

Harmadik pozíció

hexa	dec	hexa	dec
0	0	8	2048
1	256	9	2304
2	512	A	2560
3	768	B	2816
4	1024	C	3072
5	1280	D	3328
6	1536	E	3584
7	1792	F	3840

Negyedik pozíció

hexa	dec	hexa	dec
0	0	8	32 768
1	4 096	9	36 864
2	8 192	A	40 960
3	12 288	B	45 056
4	16 384	C	49 152
5	20 480	D	53 248
6	24 576	E	57 344
7	28 672	F	61 440

Ötödik pozíció

hexa	dec	hexa	dec
0	0	8	524 288
1	65 536	9	589 824
2	131 072	A	655 360
3	196 608	B	720 896
4	262 144	C	786 432
5	327 680	D	851 968
6	393 216	E	917 504
7	458 752	F	983 040

Amikor valós szegmentált címekkel dolgozunk, a szegmentált bekezdésrész a szegmensregiszterek egyikének tényleges tartalmát képviseli, és csaknem bárhová mutathat a memóriában. A relatív eltolás használata változik. A kódeltolások, amelyek a program elhelyezését mutatják, általában hexa 100-zal kezdődnek, mivel 256 (hexa 100) bájt van félretéve egyes program előtt a programelőtag számára. Az adatoknál használt adateltolások általában 0-tól indulnak. A veremeltolások általában nagy számok, mivel a verem a szegmensben belülről visszafelé működik.

A szegmentált címre egy egyszerű példa a DOS DEBUG programjának futtatása. Induláskor a „ – ” jel utasítás-promptot jelent. Egy egybetűs utasítás (**D**) hatására a DEBUG kijelzi a memória tartalmát, és a kijelzésben a bal oldalon láthatók a szegmentált címek.

Decimális-hexadecimális konverzió

A következő oldalon levő táblák mutatják be az első öt számértékhelyen levő minden egyes hexadecimális számjegy decimális egyenértékét, amely a PC-ben használt teljes tartományt lefedi. Ezeket a táblázatokat lehet használni a hexadecimális és a decimális számok közötti átváltásra.

Az alábbiakban bemutatjuk a táblák kezelését; az A1B2 hexadecimális számot használva példaként. Nézzük meg a hexadecimális szám minden egyes számjegyét a neki megfelelő helyen a táblázatban, és adjuk ösze a decimális értékeket a következők szerint:

2 az első helyiértéken	2
B a második helyiértéken	176
1 a harmadik helyiértéken	256
A a negyedik helyiértéken	40 960
Az összeg	41 394

Ugyanilyen egyszerű e táblázatok használatával a tízes számot 16-os számrendszerű számokká alakítani, de ennek leírása kissé bonyolultabb. A dolgok ismét egyszerűbbé válnak, ha egy példát mutatunk: alakítsuk át a decimális 1492-t hexa számmá! Az ötödik helyiértékre

B5. táblázat
A tízes számrendszerbeli 1492 átváltása tizenhatos számrendszerbe

Helyiérték	A legnagyobb hexa szám	A decimális érték	A decimális maradék
Az induló szám			1492
5	0	0	1492
4	0	0	1492
3	5	1280	212
2	D	208	4
1	4	4	0
Eredmény	005D4		

vonatkozó táblától indulva az első helyiértékre vonatkozó tábláig dolgozzunk. Az ötödik helyiértékben levő táblázatban keressük meg azt a legnagyobb hexadecimális számjegyet, amelynek az értéke nem nagyobb mint 1492! Írjuk le a hexadecimális számjegyet, vonjuk ki a decimális értékéből az 1492-őt, és a következő táblázaton folytassuk az új értékkel (azaz a levonás utáni különbséggel)! Menjünk tábláról táblára, amíg a maradék 0 nem lesz. Az eljárást a B5. táblázat mutatja. Az eredmény 005D4, a bevezető nullák nélkül pedig 5D4.

Hexadecimális aritmetika BASIC nyelven

A hexadecimális számokat a legkönnyebben a BASIC-kel kezelhetjük. Aktiváljuk a BASIC-értelmezőt, és vigyünk be bármilyen műveletet, amelyet végre akarunk hajtani utasítás-üzemmódban (sorszámok nélkül).

Egy hexadecimális szám – pl. a 1234 – decimális megfelelőjének kijelzésére egyszerűen a következőt tehetjük:

```
PRINT &H1234
```

Ügyeljünk minden hexadecimális szám előtt a &H előtagra, a BASIC így azonosítja a hexadecimális számokat. A decimális számok (különösen a nagy számok) legjobb kijelzéséhez a PRINT USING-formátumot használhatjuk:

```
PRINT USING "###,###,###"; &H1234
```

Egy decimális szám – pl. az 1234 – hexadecimális értékét a következőképpen jeleníthetjük meg:

```
PRINT HEX$ (1234)
```

Az eddigi példákban csak decimális és hexadecimális konstansok voltak. Csaknem ugyanilyen könnyen végrehajthatunk a BASIC-kel bizonyos aritmetikai műveleteket, és az eredményt akár a tízes, akár a tizenhatos számrendszerben megjeleníthetjük. Két példa erre:

```
PRINT USING "###,###,###"; &H1000 - &H3A2 + 16 * 3
PRINT HEX$(1776 - 1492 + &H100)
```

A változók használata a kiszámolt eredmények tárolására megkímélhet attól, hogy egy kifejezést vagy egy komplikált számot a billentyűzetről kelljen újból bevinnünk. Az olyan változókat, amelyek hexadecimális számokat tárolnak, mindig dupla pontosságú változóként (a változó nevet követő; jellel) kell írnunk ahhoz, hogy a maximális pontosságot kapjuk:

```
X = 1776 - 1492 + &H100
PRINT USING "###,###,###"; X#, 2 * X, 3 * X#
```

Hexadecimális öszeadás

Hexadecimális számok összeadásakor ugyanúgy számjegyről számjegyre haladunk, mint a decimális (tízes) számoknál. Ezt egyszerűsíti a B1. ábra, amely bármely két hexadecimális számjegy összegét kiszámolja. E táblázat használatához először keressük meg az egyik hexadecimális szám sorát, a másiknak pedig az oszlopát. A két számjegy összege a sor és az oszlop metszésénél helyezkedik el.

Hexadecimális szorzás

Hexadecimális számok szorzásánál éppúgy számjegyenként dolgozunk, mint a decimális számoknál. Ezt könnyíti meg a B2. ábra, amely bármely két hexadecimális számjegy szorzatát kiszámolja. A táblázat használatakor keressük meg az egyik hexadecimális számjegy sorát és a másik oszlopát. A két számjegy szorzata a sor és az oszlop metszésénél elhelyezkedő hexadecimális szám.

C MELLÉKLET
A karakterkészlet

Bevezetés • 358

A szabványos és a kibővített karakterkészlet • 359
A karakterformátum • 360
Az első 32 ASCII-karakter • 362
A keretrajzoló karakterek • 363
A grafikus és a blokk-karakterek • 364

Szövegállomány formátumának konvenciói • 364
Rendes szövegállomány-formátumok • 365
Szövegszerkesztő programok szövegformátumai • 366

Bevezetés

Az IBM személyiszámítógép-család 256 különböző karaktert használ. Ezeknek 0-tól 255-ig terjedő numerikus kódjaik vannak. Sokszor ezekre a karakterekre számértékükkel hivatkozunk. (A BASIC jelölés: **CHR$(0)**-tól a **CHR$(255)**-ig.) Az első 128 karakter (**CHR$(0)**-tól **CHR$(127)**-ig) valódi szabványos ASCII-karakter; az utolsó 128 karakter (a **CHR$(128)**-tól a **CHR$(255)**-ig) a különleges karakterek, amelyek a kibővített ASCII-karakterkészletet alkotják.

Általában a számítógépek a valódi ASCII-karaktereket hasonló módon kezelik, azonban van néhány eltérés az első 32 karakter kezelésében (ezen karakterek tárgyalására még visszatérünk). A számítógépgyártók határozzák el, hogy hogyan használják a 128 különleges karaktert.

Szerencsére az IBM személyi számítógépek összes típusa ugyanazt a bővített ASCII-karakterkészletet használja. Azok a számítógépgyártók, akik pontosan utánozzák az IBM személyi számítógépeket, szintén ezt használják, más számítógépgyártók azonban gyakran használnak saját karaktereket. Ezt fontos figyelembe venni, amikor más számítógépekről konvertálunk programokat, vagy amikor olyan PC-programokat írunk, amelyet szándékunkban áll más számítógépre is átírni.

0	00		32	20		64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	└	224	E0	α
1	01	☺	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	í	193	C1	┴	225	E1	β
2	02	☻	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
3	03	♥	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	├	227	E3	π
4	04	♦	36	24	$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
5	05	♣	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
6	06	♠	38	26	&	70	46	F	102	66	f	134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
7	07	•	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
8	08	◘	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
9	09	○	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
10	0A	◙	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
11	0B	♂	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
12	0C	♀	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
13	0D	♪	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
14	0E	♫	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
15	0F	☼	47	2F	/	79	4F	O	111	6F	o	143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
16	10	►	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	░	208	D0	╨	240	F0	≡
17	11	◄	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
18	12	↕	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
19	13	‼	51	33	3	83	53	S	115	73	s	147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
20	14	¶	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
21	15	§	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
22	16	▬	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
23	17	↨	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
24	18	↑	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
25	19	↓	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
26	1A	→	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
27	1B	←	59	3B	;	91	5B	[	123	7B	{	155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
28	1C	∟	60	3C	<	92	5C	\	124	7C	\|	156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
29	1D	↔	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
30	1E	▲	62	3E	>	94	5E	^	126	7E	~	158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
31	1F	▼	63	3F	?	95	5F	_	127	7F	⌂	159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

C1. ábra. Az IBM PC-család karakterkészlete
Az oszlopok a karakterek decimális, hexadecimális értékét és képernyőképét mutatják

A szabványos és a kibővített karakterkészlet

A következő BASIC-program bemutatja a 256 karaktert a numerikus kódjaikkal együtt a tízes és a tizenhatos számrendszerbeli jelzéssel is. A karaktereket a C1. ábra sorolja fel.

```
1000 ' az összes PC-karakter kijelzése
1010 '
1020 MONOCHROME = 1
1030 IF MONOCHROME THEN WW = 80 : HH = &HB000
     ELSE WW = 40 : HH = &HB800
1040 GOSUB 2000                    'a DS regiszter inicializálása
1050 FOR I = 0 TO 255              'az összes karakterkódra
1060   GOSUB 3000                  'az információ kijelzése
1070 NEXT I
1080 PRINT "Done."
1090 GOSUB 6000
1095 SYSTEM
1999 '
2000 ' initialize
2010 '
2020 DEF SEC = HH                  'a DS regiszter beállítása POKE utasításhoz
2030 KEY OFF : CLS                 'a képernyő beállítása
2040 WIDTH WW : COLOR 14,1,1
2050 FOR I = 1 TO 25 : PRINT : NEXT I
2060 PRINT "Az összes karakter bemutatása"
2070 GOSUB 5000                    'periodikus alfejezet
2080 RETURN
2099'
3000 ' a karakter kijelzése
3010 '
3020 PRINT USING " ### ";I;
3030 IF I < 16 THEN PRINT "0";
3040 PRINT HEX$(I);"   ";
3050 POKE WW * 2 * 23 + 34, I      'karakter beszúrása
3060 GOSUB 4000                    'bármilyen megjegyzés kinyomtatása
3070 IF (I MOD 16) < 15 THEN RETURN 'szünet minden egyes 16 karakter után
3080 GOSUB 6000
3090 IF I < 255 THEN GOSUB 5000
3100 RETURN
3997 '
3998 ' karaktermegjegyzések
3990 '
4000 IF I =   0 THEN PRINT         "helyközt mutat";
4007 IF I =   7 THEN PRINT         "beep (csengő)";
4008 IF I =   8 THEN PRINT         "visszalépés";
4009 IF I =   9 THEN PRINT         "tabuláció";
4010 IF I =  10 THEN PRINT         "soremelés";
4012 IF I =  12 THEN PRINT         "lapdobás";
4013 IF I =  13 THEN PRINT         "kocsivissza";
4026 IF I =  26 THEN PRINT         "szövegállomány vége";
4032 IF I =  32 THEN PRINT         "valódi üres helyköz";
4255 IF I = 255 THEN PRINT         "helyközt mutat";
4997 PRINT                         'sor befejezése
```

```
4998 RETURN
4999 '
5000 ' periodikus alfejezet
5010 '
5020 COLOR 15
5030 PRINT
5040 PRINT
5050 PRINT "Decimális – hexadecimális – karakter – megjegyzések"
5060 PRINT
5070 COLOR 14
5080 RETURN
5999 '
6000 ' szünet
6010 '
6020 IF INKEY$ < > " THEN GOTO 6020
6030 PRINT
6040 COLOR 2
6050 PRINT "Nyomjon meg a folytatáshoz egy gombot!"
6060 COLOR 14
6070 IF INKEY$ = "" THEN GOTO 6070
6080 PRINT
6090 RETURN
```

Ezzel a programmal és a mutatott karakterekkel kapcsolatban a következőket kell megjegyezni:

A program automatikusan beállítja magát monokrom vagy színes-grafikus adapterre, attól függően, hogy milyen érték van az 1020-as sorban: monokrom adapter esetén használjunk 1-t (ezt mutattuk be), ill. változtassuk az 1-et 0-ra színes-grafikus adapter vagy azzal egyenértékű adapter esetén.

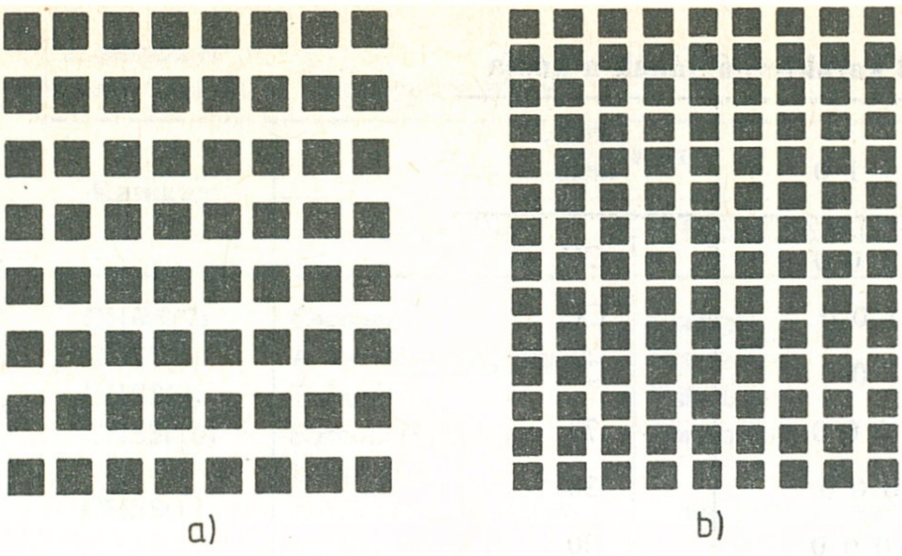
Az 1020-as sorban levő értéktől függően a program két beállítást végez el: megváltoztatja a **POKE**-utasítással a kijelzett információ tárolási helyét a képernyőmemóriában, ill. a képernyő szélességet 40 vagy 80 oszlopra állítja. A színes-grafikus adapternél a 40 oszlopos üzemmódot általában a nagyobb tisztasága miatt használják. A 3050-es sorban levő **POKE**-utasítás az, amelyik a karakterek kijelzését végzi. Erre a külön lépésre azért van szükség, mert néhány karaktert nem lehet a rendes **PRINT**-utasítással kijelezni. (☛ L. az első 32 ASCII-karakterekkel foglalkozó részben.)

A 256 különálló karakter mindegyikének megvan az egyedi megjelenése, kivéve a **CHR$(0)** és a **CHR$(255)** karaktereket, amelyek helyközként, **CHR$(32)**-ként, jelennek meg. (L. a C1. ábra).

A karakterformátum

Az összes karakter, amely a képernyőn megjelenik, egy karaktermátrixnak nevezett rácson belül rajzolt pontokból áll. Két szabványos rács van, egy a monokrom adapterre (és az anak megfelelőkre) és egy a színes-grafikus adapterre (és annak megfelelőire). Mindkét esetben a karaktereket a rácsban levő megfelelő pontok kitöltésével (megvilágításával) állítjuk elő.

A színes-grafikus adapter 8 × 8-as pontmátrixot használ, a monokrom adapter pedig egy 9 × 16-os pontmátrixot, amely egy finomabb karakterrajzolatot tesz lehetővé. A legtöbb nem-IBM PC-családtag éppúgy csak a színes-grafikus formátumot használja, mint az IBM PCjr.



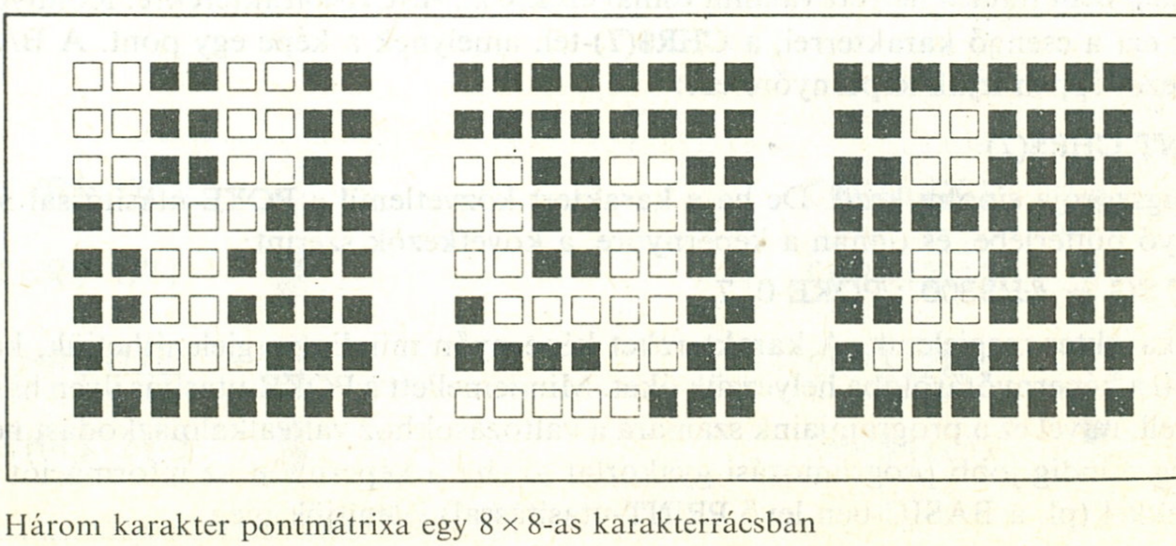
C2. **ábra.** A színes-grafikus adapter (A) és a monokrom adapter (B) által előállított pontmátrix sablon

Említésre méltó kivételek a kompakt PC-kompatibilis számítógépek, amelyek egyesítik mindkét formátumot ugyanazon képernyőn.

A mátrixnyomtatók, pl. az IBM Compact nyomtató, csak pontrácsokkal rajzolják a karaktereket. Minden egyes nyomtatótípusnak sajátos módszere van a karakterek rajzolására, és nem biztos, hogy az pontosan illeszkedik a képernyőkarakterhez. A grafikus képernyő tartalmának egy grafikus nyomtatóra való kinyomtatásának tökéletesen kell a képernyőhöz illeszkedni.

A karakterek megjelenítésének bemutatására a C3. ábrán három pontmátrixot, az *Y*-t, az *y*-t és a pontosvesszőt láthatjuk egy 8 × 8-as mátrixban.

Számos szabály van, amely a karakterrajzolásra vonatkozik. A rendes karaktereknél a két jobb oldali oszlopot nem használjuk, hogy a karakterek közötti elkülönítést biztosítsuk. Ezeket az oszlopokat csak olyan karakterek használják, amelyek a teljes mátrixot kitöltik, mint pl. a mátrixot teljesen kitöltő (**CHR$(219)**). A felső két sort az ékezetekre használjuk (a karakteréknek azon részére, amely a normál karaktermagasságon felül van). Az ékezet területét használjuk a nagybetűkre és olyan kisbetűkre, mint a *b*, a *d* és a *k*. Az alsó sort használjuk az alányúló részekre (a karakterek azon részére, amelyek a sor alá esnek), mint a *g* betűnél és az *y*-nál. Ezek az általános elvek néha kompromisszumhoz vezetnek annak érdekében, hogy jobb általános hatást érjünk el. A pontosvessző a C3. ábrán levő harmadik példánkban a vártnál egy sorral feljebb tolódik azért, hogy ne használja az alányúló sort.



C3. **ábra.** Három karakter pontmátrixa egy 8 × 8-as karakterrácsban

C1. táblázat
Az Y karakter 8 karakterbájtjának a kódja

Bit 7 6 5 4 3 2 1 0	Érték, hexa
1 1 0 0 1 1 0 0	CC
1 1 0 0 1 1 0 0	CC
1 1 0 0 1 1 0 0	CC
0 1 1 1 1 0 0 0	78
0 0 1 1 0 0 0 0	30
0 0 1 1 0 0 0 0	30
0 1 1 1 1 0 0 0	78
0 0 0 0 0 0 0 0	00

A 4-6-os és a 8-10-es grafikus üzemmódokban (a BASIC-ben az 1-6-os monitor üzemmódokban) a fenti vezérelveket használva megalkothatjuk a saját karakterrajzolónkat. A táblázatokat minden egyes karakterre 8 bájttal írjuk le. A rajz minden egyes sora 1 bájt. A bájt bitjei mutatják, hogy a sorban hol vannak pontok. Az Y karaktert pl. hexadecimálisan CC CC CC 78 30 30 78 00-ként kódoljuk. A bájtban levő egyedi biteket a C1. táblázat mutatja. (Nézzük meg közelebbről az *Y* mintát!)

Az első 32 ASCII-karakter

Az első 32 ASCII-karakternek (**CHR$(0)**-tól **CHR$(31)**-ig) két fontos felhasználása van, amelyek éppen egymással ütköznek. Egyfelől karaktereknek szabványos ASCII-jelentéseik vannak; ezeket használjuk a nyomtatóvezérlésre (pl. a **CHR$(12)** a lapdobás (form feed) karaktere és a kommunikáció vezérlésre. Az IBM használja ezeket néhány igen érdekes és hasznos képernyőkarakternek is, pl. a franciakártya-karaktereknek (szív, káró, pikk, treff a **CHR$(3)**-tól a **CHR$(6)**-ig).

Általában a számítógép összes berendezése – beleértve a nyomtatót és a képernyőt – a karakterkép bemutatása helyett valamit csinál ezekre az ASCII-karakterekre. Könnyen bemutathatjuk ezt a csengő karakterrel, a **CHR$(7)**-tel, amelynek a képe egy pont. A BASIC-ben a következőképpen írjuk képernyőre ezt:

```
PRINT CHR$(7)
```

a PC hangszórója sípolni kezd. De ha a karaktert közvetlenül a **POKE**-utasítással helyezzük a képernyő pufferjébe, és onnan a képernyőre, a következők szerint:

```
DEF SG = &HB800 : POKE 0, 7
```

akkor a karakter megjelenik. A karaktereket képernyőn mindig megjeleníthetjük, ha **POKE**-utasítással a képernyőtárolóba helyezzük őket. Mindemellett a **POKE**-utasítás ilyen használatát kerülni kell, mivel ez a programjaink számára a változásokhoz való alkalmazkodást nehezebbé teszi. Még mindig jobb programozási gyakorlat az, ha a képernyőn az információt a rendes módszerekkel (pl. a BASIC-ben levő **PRINT**-utasítással) jelenítjük meg.

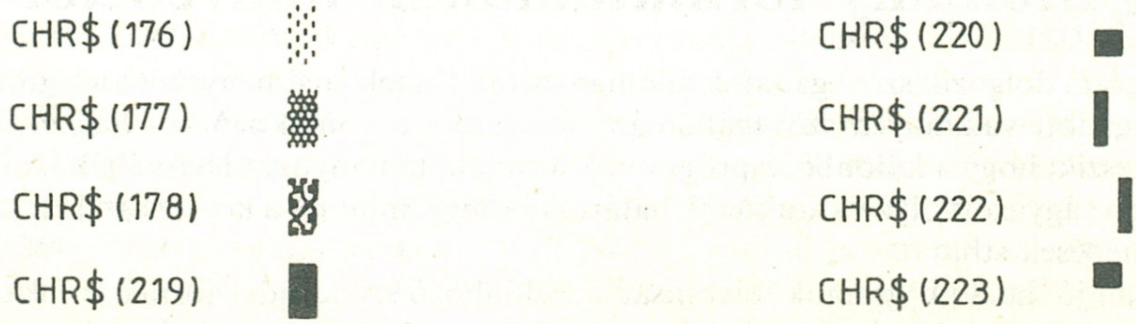
C2. táblázat
Néhány karakter képernyőre írása

Karakter	Eredmény	
	BASIC-ben	A legtöbb más nyelvben
CHR$(7)	Csenget	Csenget
CHS$(8)	A karakter megjelenik	Visszalépés
CHS$(9)	Tabulálás	Tabulálás
CHS$(10)	Soremelés és kocsivissza	Soremelés
CHS$(11)	A kurzor a bal felső sarokba mozdul	A karakter megjelenik
CHS$(12)	A képernyő törlése	A karakter megjelenik
CHS$(13)	Kocsivissza	Kocsivissza
CHS$(28)	A kurzor jobbra mozdul	A karakter megjelenik
CHS$(29)	A kurzor balra mozdul	A karakter megjelenik
CHS$(30)	A kurzor felfelé mozdul	A karakter megjelenik
CHS$(32)	A kurzor lefelé mozdul	A karakter megjelenik

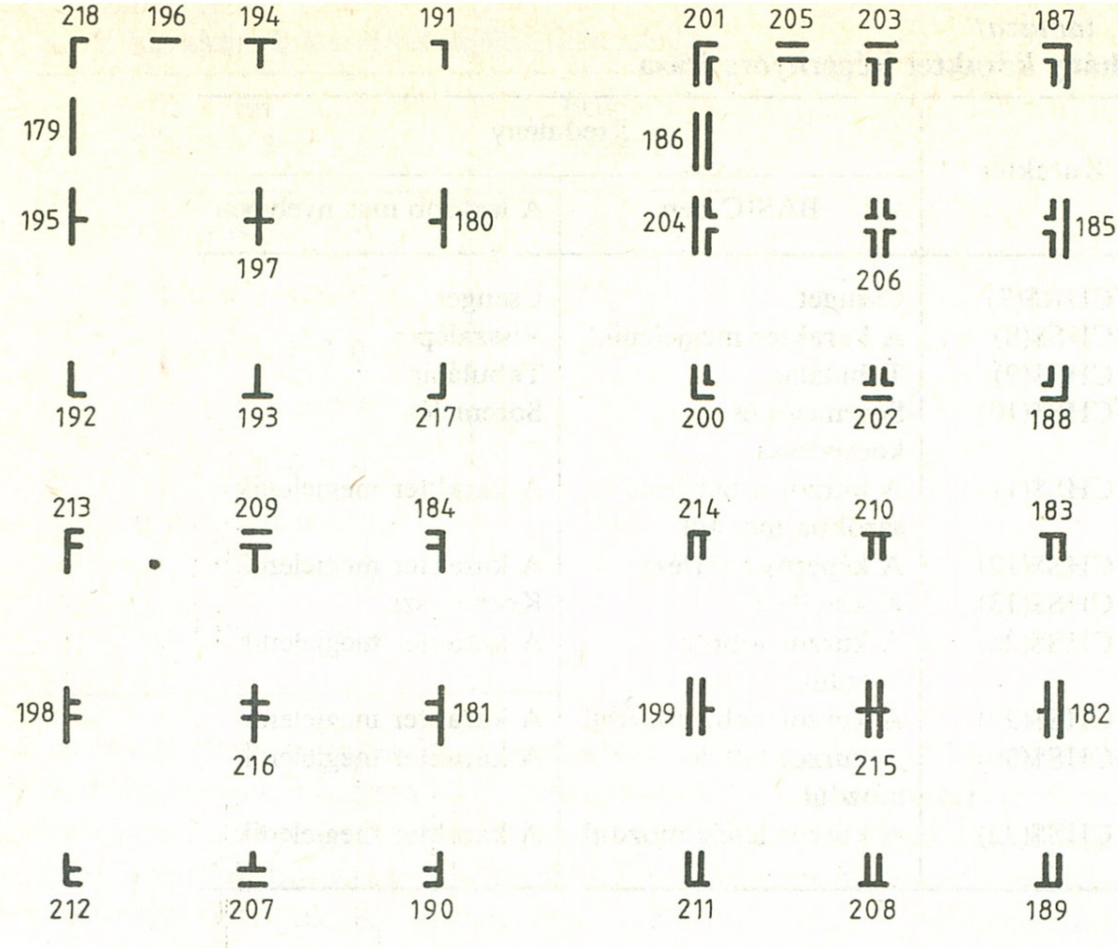
Az első 32 karakter legtöbbje megjeleníthető a képernyőn, de a képeik változhatnak attól függően, melyik nyelvet használjuk. A C2. táblázat bemutatja ezen eltérések némelyikét. A be nem mutatott karaktereket (**CHR$(0)**-tól a **CHR$(6)**-ig és a **CHR$(4)**-től a **CHR$(27)**-ig) mindig megbízhatóan előrejelezhető eredménnyel lehet a képernyőre írni.

A keretrajzoló karakterek

A különleges ASCII-karakterek közül a leghasznosabbak azok a karakterek, amelyeket egyszeres vagy dupla vonalú keretek rajzolására terveztek. Ezeknek a karaktereknek a kódjai **CHR$(179)**-től **CHR$(218)**-ig terjednek. Mivel megfelelő összeállításuk nehéz, hasznos lehet a C4. ábrán levő információ.



C4. ábra. A keretrajzoló karakterek



C5. ábra. A grafikus- és blokk-karakterek két készlete

A grafikus és a blokk-karakterek

A keretrajzoló karaktereken kívül grafikára és blokkok rajzolására két karaktersorozat használható. Az egyik sorozat négy karakter, amelyek a teljes karaktermátrixot kitöltik, de árnyalatuk különböző sűrűségű. Annál a háromnál, amelyek nem tömörek, a karakterpontok egy része be van kapcsolva, vagy az előtér színére van állítva; míg a megmaradó többi pont ki van kapcsolva, vagy a háttérszínre van állítva. A másik sorozat tartalmazza a blokk-karaktereket, amelyek színezést adnak. Ezek a karaktermátrix felét fedik. A tömörkaraktert **(CHR$(219))** használhatjuk ezekkel a félkarakterekkel.

Szövegállomány formátumának konvenciói

Számos program dolgozik szövegadatok állományaival. Ennek eredményeként a legtöbb programozó elfogadott valamilyen szövegállomány-formátumra vonatkozó konvenciót, amelyek könnyebbé teszik, hogy a különböző programok az adatállományokat használják. A formátumot a szövegbe ágyazott olyan karakterek határozzák meg, mint pl. a kocsivisszák, a soremelések, a visszalépések stb.

Általában jó, ha a programok toleránsak a különböző szövegadat-formátumokkal szemben. Fontos az is, hogy olyan programokat tervezzünk, amelyek csak kevés formátumkaraktert használva, egyszerű formátumokkal alkotják a szövegadatokat. Azokban a szövegszerkesztő

programokban, amelyeknek tudniuk kell a szöveget a képernyőn különböző formátumban megjeleníteni, nem mindig lehetséges. Először a legáltalánosabb fajtájú szövegállományt tárgyaljuk, csak utána térünk át a szövegfeldolgozó állományok tárgyalására.

Rendes szövegállomány-formátumok

A rendes szövegállomány-formátumok csak szabványos ASCII-karaktereket tartalmaznak, és nem használják a kiterjesztett ASCII-karaktereket. Az ASCII-kódtáblában az első 32 karakternek (**CHR$(0)**-tól **CHR$(31)**-ig) külön jelentése van. Néhányat közülük szövegadatok, másokat általában a kommunikáció vezérlésére használunk. Mivel ezeket a karaktereket használjuk arra, hogy a szövegformátumot vezéreljék, nem kell, hogy megjelenjenek a képernyőn; valójában az a legjobb, ha nem jelennek meg, mivel gyakran az átlagtól eltérő kép társul hozzájuk.

Csak egy maroknyi olyan formátumkarakter van, amit széles körben használnak a rendes szövegállományokban. Eredetileg olyan utasításokként alakultak ki, amelyek a nyomtatóval közölték egy kinyomtatandó oldal formátumát, és jelezték az állomány végét, Most már használatuk kiterjedt az összes kimeneti berendezésre. A következőkben minden egyes formátumkaraktert bemutatunk.

A **CHR$(26)**-ot használjuk a szövegállomány valódi végének jelzésére. Ez a karakter kerülhet az állományméret vége elé. Ez azért van, mert számos szövegfeldolgozó program az állományokat nem bájtonként, hanem nagyobb adagokban, 128 bájtonként írja és olvassa. Amikor ezzel a sebességgel adatokat ad át, a DOS csak a 128 bájtos tömb végét látja, és nem ismeri fel a **CHR$(26)** karakterrel lezárt tényleges állomány végét.

A **CHR$(13)**-at és **CHR$(10)**-et rendszerint a szövegállomány sorokba osztására használjuk, ahol az egyes sorok végét a kocsivisszával (carriage return **CHR$(13)**), és a soremeléssel (line feeld **CHR$(10)**) jelöljük, rendszerint ebben a sorrendben. A legtöbb szövegszerkesztő programnak nehézségei vannak a 255 karakterhosszt meghaladó sorokkal, és néhány a 80 karakteres sorokra korlátozódik.

A kocsivisszát önmagában is lehet használni, sajnos ezt kétféle dologként lehet értelmezni: egy sor végeként egy beleértett soremeléssel (amit a legtöbb nyomtató ad is), vagy egy visszatérésre az aktuális sor elejére, ennek következtében az egész sor felülíródik (néha a visszalépés karaktert, a **CHR$(8)**-at is használják arra, hogy a nyomtatóval felülüttesenek egy karaktert).

A **CHR$(4)**-et – a tabuláló karaktert – használjuk néha egy vagy több helyköz jelölésére a következő tabulációhelyig. Sajnos, még a tabuláció beállításokra nincs általánosan elfogadott szabály, ami a tabulátorkarakter használatát bizonytalanná teszi. A leggyakrabban használt tabulációbeállítás: a sor minden nyolcadik helye.

A **CHR$(12)** a lapdobás (form feed) egy másik formátumkarakter. Ezt a karaktert olyan nyomtatónak szóló utasításként értelmezzük, amely közli a nyomtatóval, hogy lépjen a következő oldal elejére.

Vannak más formátumkarakterek is, pl. a vertikális tabuláció, a **CHR$(11),** de a személyi számítógépeknél nincsenek olyan széles körben elterjedve.

Sok bonyodalmat kerülhetünk el, ha a programjaink a szövegadatokat egyszerű formátummal alkotják. A legegyszerűbb formátumok nem engedik meg a 255 karakternél hosszabb sorokat, és csak a kocsivissza (**CHR$(13)**), a soremelés (**CHR$(10)**) és az állomány vége (**CHR$(26)**) formátumkaraktereket használják. A legtöbb programozási nyelv, beleértve a BASIC-et és a Pascalt is, automatikusan állítják elő ezeket a formátumkaraktereket. Általában fel is dolgozzák a karaktereket úgy, hogy nekünk csak akkor kell törődnünk a szöveg formátumával, amikor kikerüljük nyelvünk rendes adatvezérlését.

Számos program (köztük a fordítók, az assemblerek) olyan szövegadatokat tud olvasni, amelynek ilyen egyszerű formátuma van. Gyakran ezek a programok nem tudnak együttműködni olyan bonyolultabb adatformátumokkal, amelyet néhány szövegfeldolgozó program állít elő.

Szövegszerkesztő programok szövegformátumai

A szövegszerkesztő programoknak a szövegformátumokban különleges igényeik vannak. Azok az állományok, amelyeket ezek a programok alkotnak, csak ritkán egyszerűek, és rendszerint a legegyszerűbb ASCII-formátumokhoz is sok egzotikus kiegészítést tartalmaznak. Általában minden egyes szövegfeldolgozó programnak megvan a maga egyedi formátumvezérlési szabálya. Szerencsére van néhány közös jellemzőjük.

E szövegszerkesztők által használt számos formátumot a kiegészített ASCII-kód – amely 128-al nagyobb, mint a rendes ASCII-kód – felhasználásával állítunk elő. Ez az egyébként rendes bájt magasértékű bitjének beállításával egyenértékű. Egy „lágy" („soft") kocsivisszát (**CHR$(141)**-et) úgy kódolunk, hogy a normál kocsivisszához (**CHR$(13)**-hoz) 128-at adunk hozzá. A „lágy" kocsivisszát gyakran használják egy próbaképpeni sorvég jelölésére, amelyet a bekezdés újraszerkesztésekor meg lehet változtatni. Másfelől a rendes kocsivisszát (**CHR$(13)**-at) használhatjuk a paragrafus végének jelzésére, amelyet nem kell változtatni. A szövegekben ilyen kódolás következtében néhány szövegszerkesztő program az egész bekezdést egy sorként kezeli.

A „lágy" elválasztások (soft hypkenation, **CHR$(173)**), amelyek a rendes elválasztásnál (**CHR$(45)**-nél) 128-al nagyobb értékűek, néha azt jelölik, hogy a szót hol lehet (sor vége esetén) szótagokra bontani. A „kemény" elválasztást szokásos karakterként kezeljük, és azt nem lehet úgy használni vagy eltávolítani a szövegtől, mint ahogy a „lágy" elválasztásokat lehet.

A normál alfabetikus szövegnek is lehet 128-as hozzáadott karakterkódja. Néhány program így jelöli a szó utolsó betűjét. A kis „a" betű pl. a **CHR$(97)**; de amikor a szó végén jelenik meg, pl. az Amerika szóban, **CHR$(225)**-ként lehet tárolni, mivel a 225 a 97 + 128 összege.

Ha a program célja sokféle szöveggel és szövegfeldolgozóval együttműködni, akkor (mint példáink mutatják) a kódolások széles körét kezelnie kell.

Tárgymutató

A, Á

ablakkezelés 152, 153
adapter
– Hercules képernyő- 62
– képernyő- 62
– monokrom 62
– professzionális grafikus (PGA) 24
– színes grafikus 62
– video- 20
adatátviteli sebesség 183
adatbusz 20,21
adatformátum 321, 322, 330, 332, 336
– halmaz 332
– integer 321, 330, 336
– lebegőpontos 322, 332, 337
adatinterfész 308
adatterület 93, 95
adattömb 187
alaptáblázat 169, 170
alárendelt szolgáltatás 139, 159
alkatalógus 100
allokációs lánc 105, 107
ANSI-billentyűzetvezérlő 344
ANSI-képernyővezérlő 344
ANSI-meghajtó 270, 343, 344
ANSI-meghajtóprogram 219
archiválási bit 102
aritmetikai művelet 17
ASCII-karakter 76, 114, 154, 358, 362
ASCII-kód 154
ASCIIZ-formátum 267
ASCIIZ-string 269
assembler 310
assembly interfész 325, 327, 334, 338
assembly nyelv 26, 141, 160, 172, 239, 310, 316, 317
AT 16, 17, 18, 19, 21, 39, 43, 66, 91, 92, 95, 106, 123, 126, 159, 162, 167, 188, 194, 195, 212, 270
attribútum 70, 154, 155
– állomány- 281
audiovonal 131
AX-regiszter 31, 37
állapot
– beszúrási 118
állapot, billentyűzetfelfüggesztési 118
– nagybetűzár 118
– záróbillentyűk 118
állapotbájt 185
állapotbit 185, 189
állomány 250, 251, 252, 253, 280, 288, 289, 292
– illeszkedő 251, 252
– random 255, 257
– soros 252
állományattribútum 281
állományfoglaltság 283
állomány foglaltsági jelző 267
állományfogó 269, 279
állományformátum 311
állományméret 256
állománymutató 280
állománynév 258
állományvezérlő tömb (FCB) 250, 260
általános célú regiszter 30, 31, 37, 139

B

base pointer (BP) 34, 38
BASIC 321
– értelmezős 317, 321
– fordítós 317, 321
BASIC-utasítás
– CALL 141
– CLOSE 107
– DEF SEG 34, 54
– INKEY$ 114, 116
– LOCATE 150
– MUSIC 128
– OPEN 107
– SCREEN 67
beépített hangszóró 126
bejelentett csomag 229
berendezéslista 191
berendezéslista szolgáltatás 205
berendezésmeghajtó 342
berendezésvezérlő 18
beszúrási állapot 118
betöltőrekord 93, 94, 97, 98
billentyű
– CapsLock 112, 118
– duplikált 114
billentyű, funkció- 115
– kurzorvezérlő 115
– különleges 115
– szerkesztő 248
– váltó 112, 177
billentyűismétlés 113
billentyűkombináció
– Ctrl-Alt-Del 112
– Ctrl-Break 113, 273
– Ctrl-NumLock 113
– Shift-PrtSc 113
billentyűmakró 110
billentyűzet 110, 249
billentyűzetbemenet 245, 247
– közvetlen 247
– tárolt 248
billentyűzetfelfüggesztési állapot 118
billentyűzetjavító program 110
billentyűzetpuffer 177
billentyűzetszolgáltatás 176, 213, 245
billentyűzetvezérlő 110
– ANSI- 344
billentyűzetvezérlő függvény 300
BIOS 24, 43, 92
bittérkép 79
blokk 22
– A. 23
– B. 23
– C. 23
– D. 24
– E. 24
– F. 24
– fej 59
– 0. 23
blokk-karakter 364
boot-strap 43
busz 12, 20
– adat- 20, 21
– cím- 20, 21
– vezérlő 20
BX-regiszter 31, 37

C

CALL-utasítás 34, 309
CapsLock 112, 118
CDUP 283

CHDIR-utasítás 277
CHKDSK-utasítás 107
ciklus 127
cilinder 91, 97, 166
cím 50, 51, 52, 53, 54, 55
– DTA- 272
– eltolási 29
– offset 29
– relatív 29, 30
– szegmens- 29, 30
– szegmentált 29, 350
címbájt 166
címbusz 20, 21
címke 101
címmegszakítás 230
CLI-utasítás 48, 112
COMMAND.COM fájl 45
compiler 310
C-nyelv 317, 336
CREAT 277
CRT-vezérlő 20, 63
CS-regiszter 32, 33, 37, 139
CX-regiszter 31, 37

CS

cserélhető szoftver 58, 59
csillapodási idő 171

D

dátum 102, 258
dátumfüggvény 304
DEBUG 97
decimális 353
decimális szám 17
DI-regiszter 34, 38
DMA-mód 170
DMA-vezérlő 18, 19, 28
DOS 92, 93, 111
DOS-függvény 45, 243, 297, 298
DOS-függvényhívás 218, 243, 244
DOS-megszakítás 218, 224, 296
DOS-partíció 96
DOS-szolgáltatás 178, 218, 299
DOS-változat 220, 272
– 1.00 220
– 1.10 220
– 2.00 220
– 2.10 220
– 3.00 220
– 3.10 220
DS-regiszter 32, 33, 37, 139
DTA-cím 272
DUP 283
duplikált billentyű 114
DX-regiszter 31, 37

E, É

EGA 24, 149
egész szám 17
egyéb függvény 305
egyéb I/O függvény 300
egyéb szolgáltatás 190, 215
előtérszín 70
eltolás 261, 262, 263, 350
eltolási cím 29
ES-regiszter 32, 33, 37, 139
EXEC 285
EX-regiszter 31, 37
értelmezős BASIC 317

F

fastruktúra 100
FAT 93, 94, 104, 106
– 12 bites 106
– 16 bites 106
fájl
– COMMAND.COM 45
– IBMBIO.COM 44, 45, 97, 101
– IBMDOS.COM 44, 45, 97, 101
– rejtett 101
– rendszer- 101
fájlnév 99
fájlnév kiterjesztése 101
FCB 250, 260
FDC 20
FDISK 96
fehér zaj 133
fejblokk 59
fejszám 166
felbontás 65
feltételjelző 35
feltételregiszter 30, 34, 38
fényceruza 151, 152
fényesség 67
FIND FIRST 287
FIND NEXT 288
fizikai formátum 91
foglaltsági kód 269
fordító 310
fordítós BASIC 317
FORMAT 97
formattálás 88, 92
formátum
– adat- 321, 330, 332, 336
– ASCIIZ- 267
formátum, állomány- 311
– fizikai 91
– lemez- 89, 90, 221
– logikai 91
– négyszeres sűrűségű 90, 91
– string 324, 330, 331, 337
– szöveg- 366
– szövegállomány- 365
formázás 165
forráskód 310
fő bájt 114, 176
fő betöltőrekord 97
funkcióbillentyű 115
függőleges szinkronjel 64
függőleges visszafutás 64
függvény
– billentyűzetvezérlő 300
– dátum- 304
– DOS- 45, 243, 297, 298
– egyéb 305
– egyéb I/O 300
– hagyományos 243
– idő- 304
– képernyővezérlő 300
– kiegészített 243, 270, 271
– kiterjesztett 267
– konzol I/O 300
– lemezkezelő 301
– memória- 306
– programvezérlő 299
– tárgymutató 304

G

gépazonosító 56, 57
gépi nyelv 26
görgetés 152, 153
grafikus karakter 154, 364
grafikus mód 62

GY

gyökérkatalógus 100

H

hagyományos függvény 243
hajlékonylemez 88, 93
hajlékonylemezes szolgáltatás 206
hajlékonylemezes vezérlő 20, 163
halmaz adatformátum 332
hang 126
hangerőszint 132
hanggenerálás 127

hanghullám 126
hangszóró 129, 130
– beépített 126
háttérszín 70
helykód 291
Hercules képernyőadapter 62
hexadecimális 353, 354, 355
hexadecimális szám 348
hiba
– időtúlfutási 184, 185, 189
– kritikus 231, 232
hibakód 226, 227, 289, 290, 291
hibakódérték 233
hozzáférési kód 279

I

IBMBIO.COM fájl 44, 45, 97, 101
IBMDOS.COM fájl 44, 45, 97, 101
IBM grafikus karakter 76
idő 102
– csillapodási 171
– késleltetési 19
– lépés- 170
időfüggvény 304
időlekérdezés 259
időszolgáltatás 193, 194
időtúlfutási hiba 184, 185, 189
időzítő 20, 126, 127, 128, 129, 130
illeszkedő állomány 251, 252
I/O társprocesszor 17
indexregiszter 30, 34, 38
indító program 42
inicializálás 42, 183
installálható berendezésvezérlő 89
installálható készülékmeghajtó 270
instruction pointer (IP) 34, 37
integer adatformátum 321, 330, 336
intenzitás 67
interfész 318, 325, 327, 334, 338
– adat- 308
– assembly 325, 327, 334, 338
– program- 308
– vezérlő- 308
interfész szubrutin 141, 142, 160, 171, 172, 239, 318
interleave 166
IN-utasítás 38
INT-utasítás 40
IRET-utasítás-40
író művelet 164

J

jelző 34
– állomány foglaltsági 267
– feltétel- 35
– kontroll- 35
– státusz- 35
JMP-utasítás 33, 34
JR 12, 19, 23, 24, 39, 43, 46, 51, 63, 66, 71, 113, 117, 131, 136, 162, 184, 192

K

karakter 154, 176, 358, 359
– ASCII- 76, 114, 154, 358, 362
– blokk- 364
– grafikus- 154, 364
– IBM grafikus 76
– keretrajzoló 363
– kész 177
– szöveges 154, 155
karakteres adat 29
karakterkiírás 158
karakter kiküldése 184
karakterlánc kiírása 159
karaktermátrix 360
karakterméret 183
karakter vétele 184
katalógus 93, 95, 98, 277, 284
– gyökér- 100
katalógusbejegyzés 99
kazettaport 186
kazettás bemenet 131
kazettás lemez 88
kazettás szolgáltatás 211
KEEP 273
keretrajzoló karakter 363
képernyőadapter 62
képernyőoldal 75, 151, 152
képernyős szolgáltatás 148
képernyőtár 63, 74
képernyőtérkép 74
képernyővezérlő függvény 300
képpont 64, 157
késleltetési idő 19
kész karakter 177
kiegészített függvény 243, 270, 271
kiegészítő bájt 114, 115, 116, 176
kiszolgáló áramkör 18
kiterjesztett függvény 267
kiterjesztett hibakód 289, 290
klaszter 95, 103, 105
kód
– foglaltsági 269
kód, forrás- 310
– hely- 291
– hiba- 226, 227, 291
– hozzáférési 279
– kiterjesztett hiba- 289, 290
– metódus- 280
– tárgy- 311
– visszaadott 267, 268
kódszegmens 234
kommunikációs szolgáltatás 182
kontrolljelző 35, 38
konzol I/O függvény 300
közvetett memória-hozzáférés 28
közvetlen billentyűzetbemenet 247
közvetlen memória-hozzáférés 19, 28
kritikus hiba 231, 232
kurzor 80, 81, 149, 150
kurzorpozíció 150, 151
kurzorvezérlő billentyű 115
különleges billentyű 115

L

lebegőpontos adatformátum 322, 332, 336
lebegőpontos szám 17
lemezegység 249, 254, 255, 274
lemezes szolgáltatás 162, 219
lemezformátum 89, 90, 221
lemezkezelő függvény 301
lemezstátusz 163
letapogatási kód 111, 112
lépésidő 170
LIFO 35
linker 311
LINK-program 313
logikai formátum 91

M

magnós szolgáltatás 186
matematikai társprocesszor 17
másolás elleni védelem 108, 167, 193
megbízhatósági teszt 42
megjelenítési mód 65, 149, 159
megnyitási mód 278
megszakítás 28, 40, 44, 45, 46, 138
– billentyűzethez kapcsolódó 119
– cím- 230
– DOS 218, 224, 296

megszakítás,
nemmaszkolható 40, 48, 113, 121
– ROM-BIOS 139
– vezérlő- 139
megszakítási szolgáltatás 193
megszakítási vektor 40, 44, 47, 48, 256, 273
megszakításkezelő rutin 231
megszakításvezérlő 18
– programozható 18
memória-áramkör 21
memóriacímzés 350
memóriafüggvény 306
memóriaszolgáltatás 206
merevlemez 88, 93, 95
metóduskód 280
méretkód 166
mikroprocesszor 12, 16
– 8088 12, 26, 27
– 80286 16
mini hajlékonylemez 88
MKDIR-utasítás 277
mód
– DMA- 170
– grafikus 62
– megjelenítési 65, 149, 159
– megnyitási 278
– színelnyomásos 69, 70
– szöveges 62
– teletype 158
monitor 65, 66
monitorkimenet 246
monokrom adapter 62, 81, 149
MOV-utasítás 33, 48
mutatóregiszter 30, 34
művelet
– aritmetikai 17
– író 164
– olvasó 164
– verifikáló 164, 165

N

nagybetűzár 112, 118
NEC-vezérlő 20
nemmaszkolható
megszakítás 40, 48, 113, 121
négyszeres sűrűségű
formátum 90, 91
numerikus adat 28
numerikus zár 112
NumLock 112

NY

nyelv
– assembly 26, 141, 160, 172, 239, 310, 316, 317
nyelv, C 317, 336
– értelmezős BASIC 317, 321
– fordítós BASIC 317, 321
– gépi 26
– Pascal 317, 330
nyomtatókimenet 246
nyomtatóspooler 228, 229
nyomtatószolgáltatás 188, 189, 214

O, Ö

offset cím 29
oldalszám 166
olvasó művelet 164
óra 20
órajel-generátor 19
óraszámláló 193, 194, 195
óraütés 128
óraütés bit 117
országfüggő információ 274, 275
OUT-utasítás 38
öntöltő 43

P

paletta 68, 156
parancsértelmező 45
paritás 183
partíció 92, 96, 97
– DOS- 96
Pascal 317, 330
PC 12, 43
PCjr 120, 121, 122
periodikus zaj 133
periódus 127
pixel 64
port 12, 20, 28, 38, 182
– soros kommunikációs 182, 183, 184, 185, 210
portszám 38, 182
PP 12
PPI 19, 127, 129
prefixum 33
professzionális grafikus adapter (PGA) 24
program
– ANSI meghajtó- 219
– billentyűzetjavító 110
– indító 42
programinterfész 308
programon belüli
kommunikációs terület (ICA) 55
programozható időzítő 19
programozható
megszakításvezérlő 18
programozható perifériaillesztő (PPI) 19, 127, 129
programszegmens 234, 257
programszegmens előtag (PSP) 234, 235, 236, 237, 238
programvezérlő függvény 299
Prokey 110

R

RAM 21, 23
RAM-lemez 88
random-állomány 255, 257
random rekordmező 256
RECOVER 107
regiszter 12, 20
– AX- 31, 37
– általános célú 30, 31, 37
– BX- 31, 37
– CS 32, 33, 37, 139
– CS- 31, 37
– DI- 34, 38
– DS- 32, 33, 37, 139
– DX- 31, 37
– ES- 32, 33, 37, 139
– EX- 31, 37
– feltétel- 30, 38
– index- 30, 38
– mutató- 30
– SI- 34, 38
– SS-32, 37, 139
– szegmens- 30, 37
– veremmutató- 34, 37
rejtett fájl 101
rekordszám 166
relatív cím 29, 30
relatív eltolás 29
rendszerfájl 101
rezidens 228
RMDIR-utasítás 277
ROM 21, 23, 42
ROM-BASIC 58
ROM-BIOS 42, 43, 45, 58, 93, 111, 112, 119, 121, 128, 138
ROM-BIOS megszakítás 139
ROM-BIOS szolgáltatás 138, 139, 140, 148, 160, 162, 198, 200
ROM-bővítés 58
rövid ugrás 178

S

sáv 88, 165
sávsűrűség 88
SETBLOCK 284
sín 12

SI-regiszter 34, 38
soros állomány 252
soros bemenet 246
soros kimenet 246
soros kommunikációs port 182, 183, 184, 185, 210
SS-regiszter 32, 37, 139
stack pointer (SP) 34, 37
státuszbájt 117, 121, 122, 163
státuszbit 177
státuszjelző 35
STI-utasítás 112
stopbit 183
string 248
– ASCIIZ- 269
stringformátum 324, 330, 331, 337

SZ

szegmens 29, 30
– kód- 234
– program- 234
szegmensbekezdés 29
szegmenscím 29,30
szegmensregiszter 30, 33, 37, 139
szegmentált cím 29, 350
szektor 88, 164, 165
szektorsorszám 225
szektorszám 166
szerkesztőbillentyű 248
szerkesztő program 311
szín 67, 70
színattribútum 154, 155
színelnyomásos-mód 69, 70
színes-grafikus adapter 62, 84, 149
színszám 68
szoftszektor 88
szoftvermegszakítás 40
szóhossz 183
szolgáltatás
– alárendelt 139, 159
– berendezéslista 205
– billentyűzet- 176, 213, 245
– DOS- 178, 218, 299
– egyéb 190, 215
– hajlékonylemezes 206
– idő- 193, 194
szolgáltatás, kazettás 211
– képernyős 148
– kommunikációs 182
– lemezes 162, 219
– magnós 186
– megszakítási 193
– memória- 206
– nyomtató- 188, 189, 214
– ROM-BIOS 138, 139, 140, 148, 160, 162, 198, 200
– soros port 210
– video- 148, 201
szögletes zárójel 37
szövegállomány-formátum 365
szöveges karakter 154, 155
szöveges mód 62
szövegformátum 366
szubrutin 309, 310, 318

T

tápvonal 20
tár 28
tárgykód 311
tárgykódú könyvtár 312
tárgymutató-függvény 304
tárolás sorrendje 32
tárolt billentyűzetbemenet 248
teletype mód 158
TI hang-IC 132, 134, 135
többtaskos üzemmód 16
tömb
– adat- 187
– állományvezérlő (FCB) 250
trigonometriai számítás 17
typematic 113

U

ugróutasítás 178
újraindítás 193
utasítás
– CALL 34, 309
– CHDIR 277
– CHKDSK 107
– CLI 48, 112
– DEBUG 97
– IN 38
utasítás, INT 40
– IRET 40
– JMP 33, 34
– MKDIR 277
– MOV 33, 48
– OUT 38
– RMDIR 277
– STI 112
– ugró 178
utasításszámláló 34, 37

V

valós szám 17
váltóbillentyű 112, 177
verem 35
veremmutató-regiszter 34, 37
veremtartalom 232
verifikáció 165
verifikáló művelet 164, 165
vezérlő
– berendezés- 18
– billentyűzet- 110
– CRT- 20, 63
– DMA- 18, 19, 28
– hajlékonylemezes 20, 163
– installálható berendezés- 89
– NEC 20
vezérlőbusz 20
vezérlőinterfész 308
vezérlőmegszakítás 139
vezérlőóra 128
videoadapter 20
videoszolgáltatás 148, 201
villogás 71
virtuális tárkezelés 16
visszaadott kód 267, 268
vizsgálj és folytasd módszer 177

X

XT 12, 39, 89, 91, 92, 95, 97, 166

Z

zajgenerátor 133

Jegyzet

495 Ft

Műszaki Könyvkiadó